A Framework for Yield Enhancement of Processor Arrays

by

Syed Shah Hadi Hussain Qadri

A Thesis Presented to the

FACULTY OF THE COLLEGE OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

COMPUTER ENGINEERING

June, 1996

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI

A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor MI 48106-1346 USA
313/761-4700 800/521-0600

A FRAMEWORK FOR YIELD ENHANCEMENT OF PROCESSOR ARRAYS

by

Syed Shah Hadi Hussain Qadri

A Thesis Presented to the
FACULTY OF COLLEGE OF GRADUATE STUDIES
KING FAHD UNIVERSITY OF PETROLEUM & MINERALS
DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the Requirements
for the Degree of

MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

Computer Engineering

June 1996

KING FAHD UNIVERSITY OF PETROLEUM AND MINERALS
DHAHRAN, 31261, SAUDI ARABIA

COLLEGE OF GRADUATE STUDIES

This thesis, written by

SYED SHAH HADI HUSSAIN QADRI

under the direction of his Thesis Advisor and approved by his Thesis Committee, has been presented to and accepted by the Dean of the College of Graduate Studies, in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER ENGINEERING

Thesis Committee

Dr. Mostafa Abd-El-Barr (Chairman)

Dr. Samir H. Abdul-Jauwad (Member)

Dr. Hasan Cam (Member)

Department Chairman

Dean, College of Graduate Studies

25/8/96

Date

Dedicated as a humble tribute to

my beloved mother

whose prayers, sacrifice, inspiration and love

led to this accomplishment

and to

the loving memory of my father

Contents

Acknowledgements i

Abstract(English) ii

Abstract(Arabic) iii

1 Introduction 1

1.1 Observations 6

1.2 Motivation 9

1.3 Objectives 11

2 Background Material 14

2.1 Manufacturing Defects 15

2.2 Fault Tolerant Processor Arrays 20

2.3 Background to the Proposed Framework 30

2.4 Concluding Remarks 35

3 Fault Distribution 36

3.1 Modeling of Manufacturing Defects 38

3.2 Effect of Defects on Yield 41

3.3 Defect Modelling 46

3.3.1 Variations in Clustering 50

3.3.2 Defect Modelling based on Clustering 51

3.4 Defect Distribution for Processor Arrays 55

3.5 Concluding Remarks 59

4 Hardware Reconfiguration Techniques for Improving Yield 61

4.1 Local Reconfiguration Schemes 63

4.1.1 Interstitial Redundancy Approach 63

4.2 Global Redundancy 65

4.2.1 Row/Column Elimination 68

4.2.2 Index Mapping Reconfiguration 70

4.3 Hierarchical Redundancy 79

4.4 Hybrid Redundancy Schemes 83

4.5 Comparison of Existing Reconfiguration Techniques 88

4.6 Concluding Remarks 90

5 The Proposed Framework 92

5.1 Overview of the Framework 94

5.2 Rules Platform - Layer 1 100

5.3 Spare Searching Techniques - Layer 2 102

5.4 Policy Based Algorithms for Sequence of Moves - Layer 3 107

5.4.1 Greedy Algorithm 109

5.4.2 Incremental Distance Algorithm 112

5.5 Formation of an Initial Solution 119

5.5.1 Pattern Based Spare Distribution 122

5.5.2 Knowledge Based Spare Distribution 131

5.6 Complexity Analysis 133

5.7 Concluding Remarks 138

6 Working with the Framework **140**

6.1 Reconfiguration using Greedy Algorithm 141

6.2 Reconfiguration using Incremental Distance Algorithm 145

6.3 Reconfiguration using Knowledge Based Spare Distribution 149

6.4 Concluding Remarks 152

7 Results and Discussion **154**

7.1 Comparison of Techniques in Different Layers 156

7.1.1 Comparison of Interconnection Resources 157

7.1.2 Comparison of Search Techniques 157

7.1.3 Comparison of Proposed Reconfiguration Techniques 160

7.2 Changes in Formation of Initial Solutions 164

7.3 Changes in Clustering Parameters 166

7.4 Variations with the Size of the Array 168

7.5 Discussion 172

7.6 Concluding Remarks 176

8 Conclusions and Future Work 177

8.1 Conclusions 179

8.2 Future Work 181

Appendix A 183

Appendix B 196

Bibliography 203

List of Figures

1.1 Different topologies for connecting processor arrays 2

2.1 A defect map obtained using a simulator. 18

2.2 Abstraction of logical array being mapped on the physical array. . . 20

2.3 Different reconfiguration approaches (a) local redundancy (b) global redundancy (c) hierarchical redundancy and (d) hybrid redundancy. . 25

2.4 The switch bus interconnection network. 26

2.5 The variable domain approach. (a) Formation of column ranges. (b) Formation of row ranges. (c) Domains of logical cells $(2,3)_L$ and $(4,4)_L$. 29

2.6 A possible initial placement of the logical array on the physical array (initial solution). 30

2.7 The reconfiguration process. 32

2.8 Final solution obtained after moves are made from the initial solution. 34

3.1 Gross yields affect whole wafers or parts of wafers. The black chips are functioning; the white chips are defective. (Scanned from [1]) . . . 43

3.2 Particle locations on "dirty" wafers show a tendency towards clustering. (Scanned from [1]) 44

3.3 Particle locations measured by laser-based particle detector in controlled environment. (Scanned from [2]). 45

3.4 Relative yield loss on a logarithmic scale. Random defects cause most of the defects. 46

3.5 Simulated defect map showing random or poisson defects. (Scanned from [2]). 48

3.6 A map generated with a negative binomial cluster generator. (Scanned from [3]). 50

3.7 A schematic showing 3 × 3 blocks on a wafer. Each block contains 2 × 3 modules. 54

3.8 Fault patterns obtained for 8 × 8 array with 15 faulty PEs and $\alpha = 1$. 58

3.9 Fault patterns obtained for 8 × 8 array with 15 faulty PEs and $\alpha = 0.3$. 59

4.1 A (1,4) interstitial redundancy array. 64

4.2 The 3-track-1-spare reconfiguration model. 66

4.3 An example of row/column elimination technique (a) (7+2)×(9+3) FTPA with faulty cells (b) Bipartite description of fault distribution. 69

4.4 Final configuration obtained using FUSS. 73

4.5 Reconfiguration of an 8 × 8 faulty array into a 7 × 7 fault free array. 76

4.6 Two level redundancy in processor arrays. 82

4.7 (The column based hybrid approach. a) Redundancy structure (b) Block organization 84

4.8 A partitioned array with spares indicated by black boxes. 87

5.1 The three layer model of the framework. 95

5.2 A simple initial solution. Square boxes show the physical locations and the logical locations are indicated in each box. (0,0) indicates a spare. 99

5.3 Compensation paths for different spare and impaired elements positions. 104

5.4 Clockwise searching of spare elements. Search begins from the elements on the left in the same row. 106

5.5 An example based on greedy algorithm. 113

5.6 An example based on incremental distance algorithm. All steps taken are with indicated manhattan distance. 118

5.7 Initial solutions for 5×5 physical array and corresponding values of accessibility. 123

5.8 Formation of X pattern based initial solution using greedy algorithm. 129

5.9 Continuation of the example based on greedy algorithm (Figure 5.8). 130

5.10 Formation of knowledge-based spare distribution. (a) Formation of variable domains (b) Partial placement obtained from bipartite matching (c) Complete placement obtained by placing cell $(2,5)_L$. 134

5.11 The final solution obtained by using greedy algorithm on 3×2 interconnection resources. 135

6.1 Plus pattern initial solution for 7×7 physical array and 6×6 logical array. 141

6.2 Reconfiguration using greedy algorithm with 3×3 interconnection resources. 142

6.3 Reconfiguration using greedy algorithm with 3×2 interconnection resources. 144

6.4 Reconfiguration using incremental distance algorithm with 3×3 interconnection resources. 146

6.5 Reconfiguration using incremental distance algorithm with 3×2 interconnection resources. 148

6.6 Row column ranges of a faulty PA. Spiral scan of physical elements and logical cells is also shown. 150

6.7 Incremental placement of logical cells on the physical array avoiding the faulty elements (knowledge based reconfiguration). 151

7.1 Comparison of survivability of 3 × 3 with that of 3 × 2 switch bus interconnection for different sizes of the array. 158

7.2 A comparison of search techniques. The two types are tree search and four-paths search. 159

7.3 Comparison of survivability of the greedy reconfiguration approach with that of incremental distance approach. 162

7.4 Comparison of survivability using plus pattern based initial solution and knowledge-based initial solution. 165

7.5 Comparison of survivability based on clustering parameter. 167

7.6 Variation in survivability of the array for increasing size of the array for incremental placement using knowledge based spare distribution. . 169

7.7 Variation in survivability of the array for increasing size of the array for greedy reconfiguration algorithm using pattern based spare distribution. 170

7.8 Variation in survivability of the array for increasing size of the array for incremental distance reconfiguration algorithm using pattern based spare distribution. 171

.1 Forward assignment and duplicate assignment. 185

.2 Explanation of the forward assignment rule. 186

.3 Effect of duplicate assignment of the cells P and R on the right-top and left-bottom cells 188

.4 Effect of duplicate assignment of the cells P and R on the right-bottom and left-top cells 188

.5 Affected cells of a forward assignment of P. 191

.6 Affected cells of a forward and duplicate assignment in 3x2 interconnection resources. 194

.7 Plus pattern initial solution. Dashed boxes indicate faulty elements. . 198

.8 Moves with distance 1. 198

.9 Moves with distance 2. 199

.10 Moves with distance 3. 199

.11 Moves with distance 4. 200

.12 Moves with distance 5. 200

.13 Move with distance 6. 201

.14 Move with distance 8. 201

.15 Move with distance 9. 202

.16 Move with distance 11. 202

List of Tables

1.1 Applications of processor arrays. 2

4.1 A comparison of array survivability for different reconfiguration schemes. 89

4.2 Comparison of interconnection requirement, fault immunity and redundancy. 89

7.1 A comparison of array survivability of 3×3 and 3×2 switch bus interconnection for different sizes of the array. 158

7.2 A comparison of array survivability of tree and four-path search of suitable spare for different sizes of the array. 160

7.3 A comparison of array survivability of greedy algorithm and incremental distance algorithm. 163

7.4 Comparison of array survivability using plus pattern based initial solution and knowledge-based initial solution. 166

7.5 Comparison of survivability based on clustering parameter. 168

7.6 Variation in survivability of the array for increasing size of the array for incremental placement using knowledge based spare distribution. . 170

7.7 Variation in survivability of the array for increasing size of the array for greedy reconfiguration algorithm using pattern based spare distribution. 170

7.8 Variation in survivability of the array for increasing size of the array for incremental distance reconfiguration algorithm using pattern based spare distribution. 171

7.9 A comparison of array survivability of proposed techniques. 174

.1 ϕ^r's and effective ϕ^r's of passive cells in V-BAR (A,P) in Figure .5. . 193

ACKNOWLEDGEMENT

First and foremost, all praise and thanks to Allah, *subhanahu-wa-ta-Aala*, the Almighty, Who gave me the opportunity, strength, determination and patience to complete this work successfully. I seek His mercy, favor, and forgiveness, and peace and prayers be upon his Prophet. Acknowledgements are due to King Fahd University of Petroleum & Minerals for having given me an opportunity to pursue my graduate study here, for providing research facilities and financial assistance to me during the course of the MS program.

I am extremely thankful and deeply indebted to my thesis committee chairman, Dr. Mostafa Abd-El-Barr for his invaluable help and advice. I acknowledge him for his untiring efforts, constant encouragement, unlimited support, concern, patience, guidance and valuable time during all the stages of this work and above all for having complete faith in me.

I wish to express my sincere appreciation and gratitude to my thesis committee members Dr. Samir Abdul-Jauwad and Dr. Hasan Cam for their interest, constructive criticism and personal understanding. I acknowledge them for their useful and valuable comments and suggestions.

I am also thankful to the Dean of college of computer sciences, Dr. M. S. T. Benten and other faculty members for their cooperation, throughout the duration of my stay in KFUPM.

I sincerely appreciate the help and guidance provided by my dear friend late Mr. Ashrafuddin and Mr. Amer Mahmood, whose untimely deaths had a great impact on me, may Allah forgive his sins and grant him a place in Paradise.

I would also like to thank the secretaries of the COE Dept., Mr. Hafeez Moghul and Mr. Khurshid for their help throughout my stay in KFUPM. I express my thanks to my friends Waris, Mohsin Nadeem, Imtiaz, Ilyas, and Mansoor for their help. Special thanks are due to Mr. Nisar-Ul-Haq, Mr. Khaja Naseer and Mr. Feroze Daud for their help in programming.

Finally, without the emotional and moral support, prayers, sacrifices, love, patience, encoragement and understanding of my mother, brothers and other family members the completion of this work would not have been a possibility. This work is dedicated to my mother for taking pains to fulfill my personal and academic pursuits and shaping my personality and to the loving memory of my father whose dream was to see me reach the pinnacle of my academic career.

THESIS ABSTRACT

Name: SYED SHAH HADI HUSSAIN QADRI

Title: A FRAMEWORK FOR YIELD ENHANCEMENT OF PROCESSOR ARRAYS

Degree: MASTER OF SCIENCE

Major Field: COMPUTER ENGINEERING

Date of Degree: JUNE 1996

Techniques for fault tolerance together with redundant circuits have been used to increase the manufacture yield and productivity of integrated-circuit chips. Spare processing elements, interconnection resources (switches and links) along with reconfiguration algorithms are used for yield/reliability enhancement in processor arrays (PAs). Yield falls sharply with incresing area. Therefore, it is desirable to optimize spare redundancy and interconnection resources used for reconfiguration. This can be done if yield estimates for different combinations of spare and interconnection redundancy are obtained. To obtain yield estimates there is a need for an integrated tool which can evaluate survivability (a measure of yield) for different combinations of spare and interconnection redundancy. An efficient hardware reconfiguration approach is the rule-based reconfiguration technique [4]. In this study, we propose a framework that allows us to investigate various reconfiguration possibilities using rule-based approach. The framework is constructed in three layers. The first layer provides a platform to check validity of the placement of functional cells. The first layer can be efficiently used to reconfigure through congestion avoidance using 3×3 *and* 3×2 *switch bus interconnection. The second layer provides utilities to search suitable spares for reconfiguration. In the third layer sequence of spare impaired element substitutions is formulated in the form of reconfiguration techniques. The framework provides a comprehensive environment which can be used to evaluate survivability with different types of interconnection resources. Chip designers can use the framework to predict yield for different combinations of spare and interconnection redundancy.*

Keywords: Yield, Processor Arrays, Fault-Tolerance, Reconfiguration and Framework.

King Fahd University of Petroleum and Minerals, Dhahran.

June 1996

خلاصة الرسالة

اسم الطالب الكامل : سيد شاه هادي حسين قادري

عنـــوان الدراســـة : هيكل لتعزيز الإنتاجية في مصفوفة العالج

التـخـــصـــص : هندسة الحاسب الآلي

تاريـــخ الشهـــادة : يونيو (حزيران) ١٩٩٦م - صفر ١٤١٧هـ

لقد تم استعمال اساليب لتحمل الاخطاء مع استخدام دوائر زائدة عن الحاجة لزيادة الانتاجية المصنعة في رقائق الدوائر المتكاملة. كما استعملت عناصر معالجة احتياطية، وروابط مصدرية (مفاتيح تحويل، وصلات) بجانب خوارزميات إعادة الوضع بغرض تعزيز الانتاجية/الموثوقية في المعالجات المصفوفة (PAs). إن الانتاجية تتناقص بحدة عندما تزداد المساحة، لذا فإنه من المرغوب أن يستحسن احتياطي الوفرة والروابط المصدرية المستعملة في اعادة الوضع. وهذا ممكن عمله إذا حُصِّل تقديرات للانتاجية لمجموعات مختلفة من احتياطي وروابط الوفرة. ولإحراز تقديرات للانتاجية هناك حاجة لوسيلة متكاملة تستطيع تقدير النجاة (مقياس انتاجية) لمجموعات مختلفة من احتياطي وروابط الوفرة. ويعتبر اسلوب اعادة الوضع المبني على القواعد طريقاً فعالاً اعادة وضع (الهاردوير) [٤] . وفي هذه الدراسة اقترحنا هيكلاً يمكننا من تحري امكانيات مختلفة اعادة الوضع باستخدام اسلوب اعادة الوضع المبنى على القواعد وقد بُني الهيكل من ثلاث طبقات. الطبقة الأولى تزوّد الارضية لفحص صحة اماكن الخلايا الوظيفية. ويمكن استعمال هذه الطبقة بفعالية لاعادة الوضع من خلال تجنب الازدحام باستعمال ٣×٣ و ٣×٢ رابطة مفتاح تحويلي. أما الطبقة الثانية فتوفر مرافقاً لبحث احتياطيات ملائمة لاعادة الوضع. وفي الطبقة الثالثة فإن سلسلة احتياطي التالف من تعويضات العنصر تتشكل على هيئة اساليب اعادة الوضع. ويوفر هذا الهيكل بيئة شاملة يمكن استخدامها لتقييم النجاة باشكال مختلفة من الروابط المصدرية. ويستطيع مصممو الرقائق استخدام هذا الهيكل لتوقع الانتاجية لمجموعات مختلفة من احتياطي الوفرة والروابط المصدرية.

درجة الماجستير في العلوم

جامعة الملك فهد للبترول والمعادن

الظهران، المملكة العربية السعودية

يونيو (حزيران) ١٩٩٦م - صفر ١٤١٧هـ

Chapter 1

Introduction

High-performance parallel processing architectures have become quite common with the advancement in very large scale integration (VLSI). Processor arrays represent one such architecture. The input/output bandwidth of these architectures is very high. The array consists of a large number of identical processing elements (PEs) connected locally. Since the processing elements are identical, any processing element in an array can perform the functions of any other processing element in the array.

Some commonly used topologies of processor arrays are shown in Figure 1.1. A linear array is a one-dimensional array, with low input/output rate and simple operation. Two dimensional arrays include the square array, the hexagonal array and the tree array. The input/output bandwidth is high and the operation is more complex.

Processor arrays are used for different types of applications. They are widely used

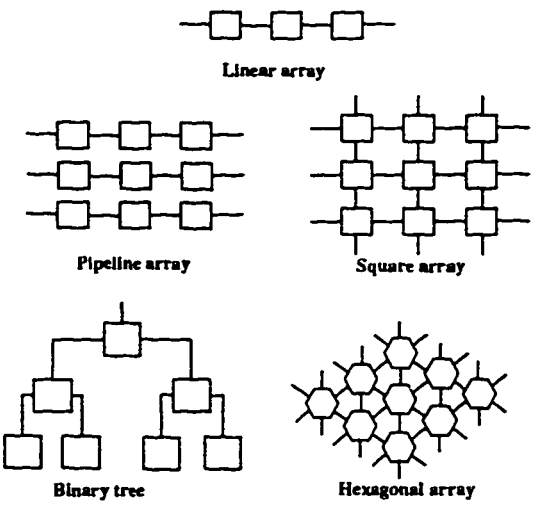


Figure 1.1: Different topologies for connecting processor arrays

in digital image and signal processing, dynamic programming, matrix arithmetic, sorting and searching. Different topologies and their domain of applications are tabulated in Table 1.1.

Array structure	Computation function
1-D Linear array	FIR-filter, convolution DFT.
2-D Rectangular array	Graph-algorithms, dynamic programming, DFT, matrix arithmetic.
2-D Hexagonal array	Matrix arithmetic
Tree structure	Sorting and searching
Multiple pipeline	Vector operation.

Table 1.1: Applications of processor arrays.

The most important factor which determines the cost of production of VLSI/WSI

circuits is *yield*. *Yield* can be defined as the the average fraction of good chips delivered by a production line [5].

There are two types of defects in VLSI circuits which are relevant for yield analysis. They are *gross area defects* and *random defects*. Gross area defects are caused by manufacturing errors that make parts of a wafer or entire wafers faulty (improper functioning). Random defects results from photo defects, pinhole defects, leakage defects, etc. and are distributed throughout the wafer causing scattered faults throughout the array. Gross yield loss accounts to about 4% of total faults [6] and [7]. Therefore, defect modeling in VLSI/WSI circuits concentrates mainly on modeling defects due to random defects. It is expected that if defects are modelled using large area negative binomial model for modeling faults in processor arrays, then more realistic estimates of yield are obtained [8].

It is well recognized that yield in large VLSI and WSI circuits is very low. It is possible to improve the yield of VLSI circuits by including redundant spare modules (hardware) on the chip. These spare modules can be used to replace faulty modules at the manufacture time, thus improving the yield. Processor arrays have regular structures and all modules in the array are identical. Therefore, spare modules can be added easily. A number of techniques can be used to improve the yield of processor arrays through hardware redundancy. They are classified into three categories in [9]. They are local reconfiguration, global reconfiguration, and set switched

(hierarchical) reconfiguration. A combination of hierarchical and global reconfiguration techniques called hybrid approach is presented in [10]. All these techniques use hardware redundancy in the form of spare processing elements and switching mechanism for reconfiguration. The switching mechanism can be a switch element, a switch bus, or a switch network [9].

In local reconfiguration techniques, faulty elements are directly replaced with spare elements. Therefore, for a group of elements in the array there should exist a spare which can directly replace a failed module [11] and [12]. In global reconfiguration techniques, redundant elements are shared by faulty elements in the whole array for reconfiguration [13] [4] and [14]. Set switched or hierarchical redundancy combines the local and global reconfiguration techniques. The array contains equal sized subarrays. There is redundancy within a subarray and there are spare subarrays. The subarrays are individually reconfigured first using local reconfiguration. Then the complete array is formed from those subarrays which have been reconfigured successfully [15]. The hybrid approach is a slight deviation from the hierarchical approach with global redundancy in both levels. Instead of using dedicated spares in each block, the spares are shared between neighboring blocks. Since the spare elements are shared, we cannot switch blocks as in hierarchical reconfiguration approach. Therefore, spare blocks are not used in this approach [10].

Many spare placement schemes and switch and track models have been proposed

for improving yield of arrays, through hardware redundancy. A switch and track model contains a fabric of interconnected switches and processing elements with metallic tracks. The function of each switch is to re-route signals on metallic tracks. This can be done by using either soft switches or hard switches. Soft switches use multiplexers and other transistor based circuits to enable flow of signal in the desired direction. Hard switches use laser programmable fuses to make or break connections between metallic tracks [16].

Using soft or hard switches we can have different types of switching mechanisms. The switching mechanism depends on the type of reconfiguration approach used. Local reconfiguration schemes require switches and tracks to directly replace faulty PE (PEs) with spare PE (PEs) in a group of PEs. Therefore, switching is performed locally within each group of PEs. In global reconfiguration schemes PEs are embedded in a fabric of interconnected switches. The switch fabric is shared by all the PEs. One of the most widely used switch fabric is called switch bus interconnection [13] [4] [17].

Depending on the number of switches and tracks, switch bus interconnection fabric can be classified as 3×3, 3×2, 2×3 and 2×2. In 3×3 there are three tracks in each of the channels between columns of PEs and three tracks in each of the channels between rows of PEs. The number of tracks in the channels are similarly defined for 3×2, 2×3 and 2×2 switch buses.

Hybrid reconfiguration schemes use the same fabric used for global reconfiguration.

In hierarchical reconfiguration schemes two types of switches are used. One for local reconfiguration within a subarray and the other for switching subarrays.

1.1 Observations

A prominent problem in WSI is that there is excessive loss of yield due to defects. Therefore, WSI has to use redundancy to make up for the excessive loss of yield that result due to defects [2]. In processor arrays, WSI uses redundancy in two forms, spare redundancy and switching redundancy. Depending on the assumptions made on average number spare PEs, the number of wires in the inter-module interconnection area, and quantity of switching modules used, the effect on the yield improvement will vary [18].

However, redundancy cannot be added boundlessly. It has been observed that in practice yield falls sharply with increasing chip area [19] and yield improvement saturates above a limited amount (10% to 15%) of redundancy [17] and [18]. A large amount of interconnection resources, on one hand may achieve higher probabilities of reconfiguring faulty arrays, but on the other hand require more hardware overhead and thereby increase susceptibility to faults [20]. These factors indicate that the chip has to be designed around an optimal amount of additional logic on the chip to improve the yield [18].

Local reconfiguration techniques require very large amount of redundancy (at least 20%) and have limited capacity to tolerate faults. From the redundancy ratio indicated in [17] and [18], yield loss may be too high. Therefore, it is expected that local sparing may not be cost-effective.

Hierarchical reconfiguration schemes require moderately high spare redundancy as well as interconnection redundancy, especially for switching blocks. Global reconfiguration techniques requires lesser percentage of spare redundancy as compared to other hardware reconfiguration techniques and have high survivability. However, the optimal interconnection requirement to successfully reconfigure arrays under different possible fault distributions is 3×3 [21].

It is reported in [18] that when yields are dominated by interconnection area, yield improvement is not large. Therefore, it is not certain whether 3×3 interconnection switch fabric is cost effective or not. This can be evaluated only when such circuits are manufactured on commercial fabrication cites.

An important figure of merit which relates to the yield of processor arrays is *survivability.* It is the probability with which a faulty array can be configured, given the number of faulty cells in the array.

Global reconfiguration techniques have very high survivability and require lesser percentage of spare redundancy as compared to other hardware reconfiguration techniques. A 3×2 interconnection switch fabric has been used for global reconfiguration, but figures of survivability are lower than those obtained for 3×3 [21] [22]. One

reason is that generic rules like those available for 3 × 3 have not been applied to 3 × 2 interconnection switch fabric. By modifying the rules for 3 × 3 switch bus we can make the complete set of rules for 3 × 2 switch bus interconnections (see appendix A).

Due to the existence of limits on maximum and minimum redundancy required for manufacturing, there is an optimal spare redundancy and interconnection redundancy combination which maximizes yield. The optimal combination depends on size of the array and size of the processing elements. Since array size and processing element size are not homogeneous, the optimal combination of redundancy is application dependent. If acceptable yields can be obtained with the use of lesser interconnection resources then there can be a substantial improvement in yield due to lesser area. Lesser interconnection resources will also require lesser control lines (signals) and lesser time to reconfigure the array.

To test different combinations of spare redundancy and interconnection redundancy a framework is needed which is generic and can allow evaluation of survivability using faults generated by large-size negative binomial fault model [1]. Such framework should be flexible to allow experimentation with different reconfiguration techniques using different amounts of spare redundancy and using different switch fabrics.

Due to the complexity of reconfiguration using different switch bus fabrics, very few techniques have been applied to more than one type of switch bus fabrics. Spare redundancy is also fixed in some of the reconfiguration approaches. The main reason

which makes it difficult to apply different reconfiguration approaches using different amounts of spare redundancy on various switch fabrics is the lack of a mechanism which validates routability of the placement of functional elements.

1.2 Motivation

Global reconfiguration techniques use lesser redundancy and achieve higher probability of survival. Architectures which support global reconfiguration techniques require more interconnection resources and allow complex interconnection patterns to improve survivability. One of these architectures which is very widely used is the switched bus interconnection fabric.

Switched bus interconnection fabric comprises of two disjoint planes of switches and links: horizontal plane and vertical plane. In the horizontal plane data flows from left to right while in the vertical plane data flows from top to bottom. Isolation of the two planes simplifies the reconfiguration algorithm considerably.

In each plane we can have 3, 2 or 1 track per row/column. If there are 3 tracks per row and 2 tracks per column then a 3×2 switched bus interconnection architecture is formed.

As a consequence of the separation of the two planes there is an increase in the amount of reconfiguration resources required for the whole array. Therefore, differ-

ent amounts of reconfiguration resources have been proposed and tested. Techniques which have used 1×1 switched interconnection achieve low survivability [23]. Techniques which have high survivability use 3×3 interconnection resources [13] [4]. Therefore, there is a need for a framework which allows us to experiment with 3×3 and 3×2 interconnection resources. To make a framework which allows estimation of probability of survival on different interconnection architectures, we should build a platform in which rules for 3 tracks and 2 tracks per row/column are implemented. Such platform allows experimentation on 3×3, 3×2, 2×3 and 2×2 interconnection architecture.

The other optimization criterion is spare redundancy. Different reconfiguration techniques use different amounts of spare redundancy on switched bus interconnection resources [23] [22] [13] [24] [4] and [10]. Efficacy of different reconfiguration mechanisms is dependent on spare redundancy used. Therefore, the framework should be flexible to incorporate different levels of spare redundancy and allow experimentation with some of the reconfiguration mechanisms.
To allow incorporation of different percentages of redundancy, we can start the reconfiguration from an initial solution. The initial solution with the desired spare arrangement can be obtained by using the existing reconfiguration techniques.

To support freedom of movement of functional cells within its neighborhood, in

any desired direction, the rule based approach proposed by [4] can be used effectively. This approach gives us the flexibility to ripple in any direction, provided the rules for reconfiguration are satisfied. Moreover, the proposed rules for 3×3 [4] can be modified to obtain rules for 3×2, 2×3 and 2×2 interconnection resources. In this thesis, we propose the use of the rules proposed by [4] for reconfiguration of processor arrays.

Using the rules for 3×3 and 3×2 interconnection resources, we should construct an underlying base (platform), on which the logical cells can be moved from one valid location to another valid location in the physical proximity. The platform for movement of logical cells should provide a flexible environment to experiment with a variety of reconfiguration techniques and interconnection resources.

1.3 Objectives

To make a framework for yield enhancement of processor arrays it is necessary to simplify the task of reconfiguration. The reconfiguration of faulty processor arrays can be divided into three tasks. First is to avoid congestion in any region of the switch fabric. Second is to substitute faulty PEs by spare PEs. Third is to choose a sequence of substitutions of faulty PEs with spare PEs.

The framework for yield enhancement of processor arrays has been modeled in three layers to tackle the different tasks. The first layer forms the core and provides a rules platform which validates routability of signals on different switch fabrics. Using the platform we can reconfigure the array by avoiding placements which can cause congestion in some of the channels. The rules platform supports 3×3, 3×2, 2×3, and 2×2 switch bus fabric for interconnection.

The second layer deals with the problem of substituting a faulty PE with a suitable spare PE. More interconnection resources are required to reconfigure a faulty PE which is further away from a spare PE. Another factor is that when a spare PE is substituted then it is not available for other faulty PEs. Therefore, a good choice has to be made with regards to the substitution of a faulty PE.

The third layer deals with the problem of ordering the substitution of faulty PEs. As more substitutions are made lesser interconnection resources are left on the switch fabric for reconfiguration of other faulty PEs. If all reconfiguration resources around a faulty PE are used, then the faulty PE cannot be substituted with a spare PE even though some unused spare PEs exist in the array. Therefore, the substitutions have to be arranged in an order which prevents congestion globally.

Using the three layer model, different combinations of spare redundancy, switch fabrics and reconfiguration techniques can be experimented with.

In this work we propose to experiment with 3×3 and 3×2 interconnection resources. The results will be useful for chip designers to deduce the feasibility of 3×2 switch

bus.

As the size of the array increases, it is expected that survivability falls. We propose to evaluate degredation in survivability for large sized processor arrays. These results will be useful for chip designers to evaluate feasibility of manufacturing large size PAs.

The rest of the thesis is organized in the following manner. In Chapter 2 we present relevant background material. Chapter 3 describes fault modeling for distribution of faults. Available techniques for yield improvement are described and a comparison of their performance is given in Chapter 4. Chapter 5 describes the proposed framework. We explain the use of framework with relevant examples in Chapter 6. In Chapter 7 we report results and conclude in chapter 8 with possible future extensions to this work. Appendix A explains the formulation of rules for 3×3 and 3×2 interconnection resources. In Appendix B we illustrate reconfiguration of a large size PA using a proposed technique.

Chapter 2

Background Material

The primary motive for introducing fault tolerance in VLSI/WSI circuits is *yield enhancement*, i.e. increasing the percentage of fault-free chips obtained. The active area of monolithic VLSI chips has always been limited by the random fabrication defects, which appear impossible to eliminate in even the best manufacturing processes. The larger the circuit, the more likely it will contain such type of defects and fail to operate correctly. Therefore, larger circuits demand a fault-tolerance capability to overcome fabrication defects.

Fault tolerance for yield enhancement of processor arrays is under study for a long time. To tolerate manufacturing defects extra processing elements and interconnection resources are used. When some of the functioning PEs are found to contain faults then the array is restructured to replace faulty PEs with spare PEs.

In this chapter we introduce the relevant terminology used in connection with yield

enhancement of processor arrays. Section 2.1 deals with manufacturing defects. Section 2.2 introduces the terminology used in fault tolerant processor arrays. In Section 2.3 we present background to the proposed work. In Section 2.4 we present some concluding remarks.

2.1 Manufacturing Defects

Spot defects are random local defects from materials used in the process and environment causes, mostly unwanted chemical and airborne particles deposited during the various steps of the process. Some spot defects might cause missing patterns or open circuits, while others cause extra patterns or short circuits. Examples include missing metal (or diffusion or polysilicon), extra metal (or diffusion or polysilicon), missing vias between two metal layers or contacts between a metal layer and polysilicon, and shorts between the substrate and metal (or diffusion or polysilicon) or between two separate metal layers.

Definition 2.1 *Defect* is defined as any physical anomaly on a chip (wafer) which may or may not cause improper functioning of a VLSI (WSI) circuit. □

Not all spot defects are structural defects resulting in discrete faults such as line breaks and short circuits. A defect causes a discrete fault only if it is large enough

to connect two disjoint conductors or disconnect a continuous pattern.

Some defects that do not cause structural faults can result in parametric faults, where the electrical parameters of some devices lie outside the desired limits of operation, affecting the performance of the circuit. Therefore, we define a fault as follows.

Definition 2.2 *Fault* on a VLSI/WSI wafer is a defect which causes improper functioning of the circuit. □

Existence of faults on a chip will make the chip function improperly. The number of faults that occur on a chip and the frequency with which they occur (in a lot of chips) is very important to manufacturers. If more chips (wafers) are damaged due to defects, then the overall cost of production will increase as the faulty circuits have to be discarded. To measure the effects of faults in VLSI/WSI circuits, the most important parameter used is *yield.*

Definition 2.3 *Yield* can be defined as the average fraction of good chips delivered by a production line. □

Percentage yield is the percentage of good chips delivered on the production line. Faults due to spot defects appear in discrete form scattered throughout the array. Faults appearing on one chip (wafer) may differ from the faults that occur on other

chips (wafers). Therefore, it is necessary to determine the average number of circuit faults. This can be predicted from the number of defects that exist on a chip (wafer). The average number of manufacturing defects of type i (for example photolithographic defects of the open circuit type) is usually described by its defect density D_i.

Definition 2.4 *Defect Density of type i* is the average number of defects of type i per unit area. □

Using the defect density, the average number of defects on a chip (wafer) λ_i of defect type i can be evaluated by using the equation

$$\lambda_i = D_i A$$

where A is the total area of the chip (wafer). The average number of defects λ is the average of all defect types.

Definition 2.5 *Average number of defects* λ, on a chip (wafer) is defined as the average number of all defects types that occur on a chip (wafer). □.

Though defect density is a good measure for obtaining yield estimates, it is not sufficient to make a defect distribution. This is because the probability that a defect will cause a fault might depend on the exact geometrical position of the defect. Therefore, it is necessary to consider the locations of these defects on the

wafer. It is confirmed that defects tend to group together on VLSI/WSI circuits. The tendency of grouping is termed as clustering and it is defined as follows.

Definition 2.6 The tendency of defects to occur close to each other showing dependence on their relative locations is known as *clustering.* □

A *cluster* is a region which encloses all the defects which demonstrate dependence on each others, location. A hypothetical defect map obtained by using a simulator is shown in Figure 2.1. There may be more than one cluster source on a chip.

Having all the defects closely clustered in VLSI/WSI circuits causes damage in a

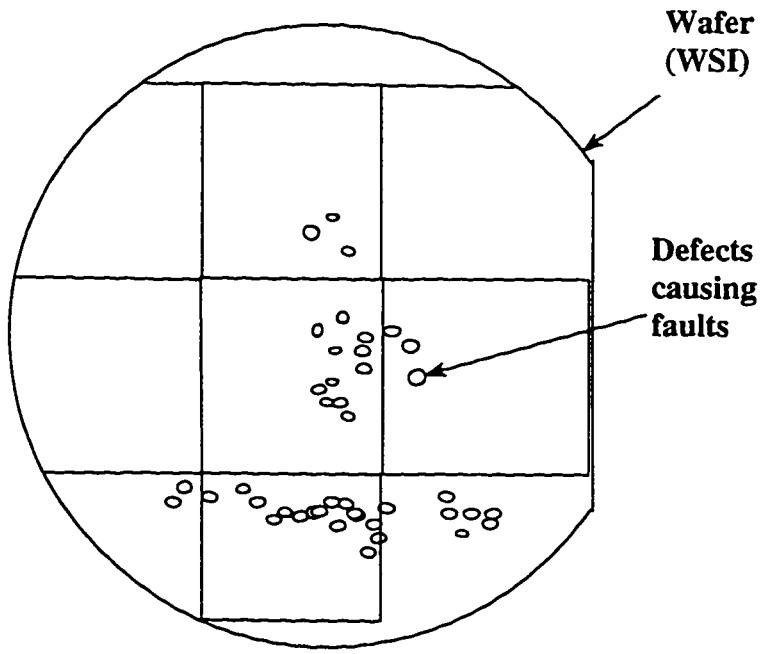


Figure 2.1: A defect map obtained using a simulator.

small region on the VLSI/WSI circuit. The effect of the defect map shown in Figure

2.1, when superimposed on a processor array is shown in Figure 2.2. Since some processing elements are faulty the processor array is faulty.

Definition 2.7 A processor array is a **faulty array** (FA) if some of its processing elements are faulty.□

If the defects are randomly distributed then faults are expected to occur throughout the array. If the same number of defects are scattered throughout the array then almost all the PEs will be defective as a single fault on a PE, causes the PE to become faulty.

The defect map shown in Figure 2.1 shows more than one defect sources. If the wafer is divided by a grid as shown in the figure then there are six defect sources each enclosed within a *quadrant* (also called *window*) of the grid. If all the defects were to be part of one cluster within a quadrant then lesser PEs may have been faulty. It is possible to increase or decrease the size of a quadrant to enclose faults. Depending on the location of faulty elements on the processor arrays clustered fault distribution of PEs in processor arrays can be classified into two types: semi-clustered and heavily clustered.

Definition 2.8 A distribution of faults on a VLSI/WSI circuit is said to be semi-clustered if there is more than one cluster of faulty PEs in a processor array. □

Definition 2.9 A distribution of faults on a VLSI/WSI circuit is said to be heavily-clustered if there is one cluster of faulty PEs in a processor array. □

In Figure 2.2 there is one cluster of faults. Therefore, it is heavily clustered. However, if PE (4,3) were not faulty then there would have been two clusters. The fault distribution in this case would be semi-clustered.

2.2 Fault Tolerant Processor Arrays

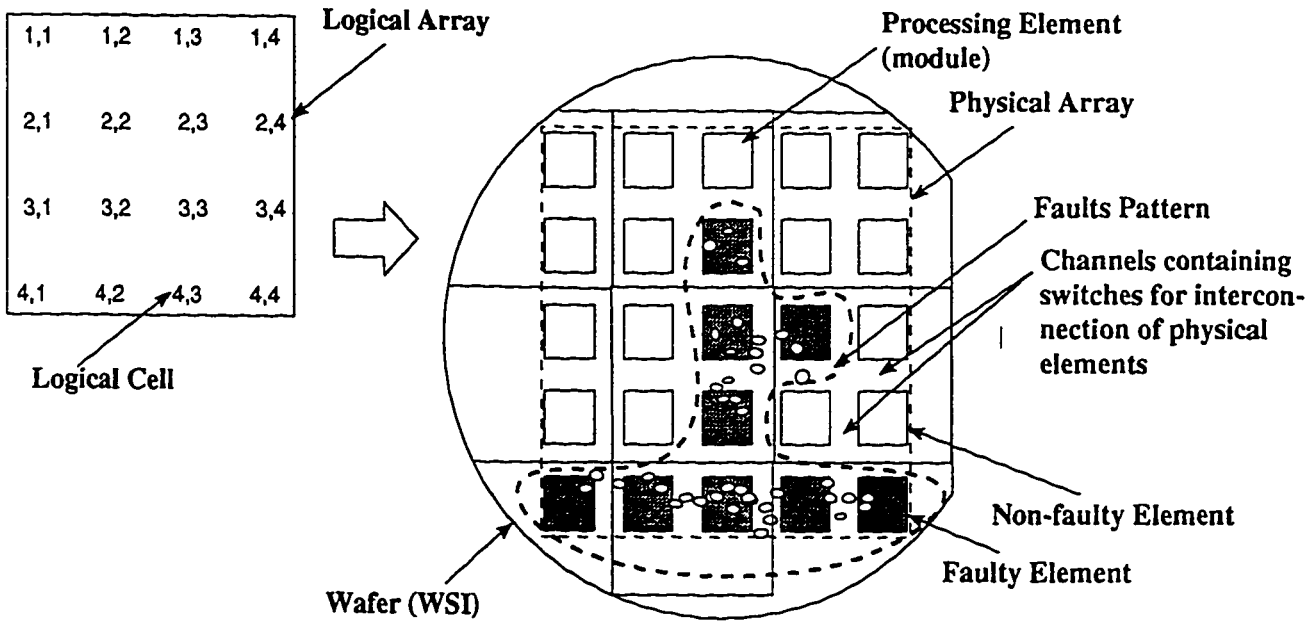


Figure 2.2: Abstraction of logical array being mapped on the physical array.

To tolerate different kinds of faults that appear on processor arrays many techniques have been proposed. These techniques can be classified as local reconfiguration techniques, global reconfiguration techniques, hierarchical reconfiguration

techniques and hybrid reconfiguration techniques. In all reconfiguration schemes an attempt is made to form a functional array from a faulty array. The size of the physical array is larger than the size of the logical array. The extra processing elements provided in the array are redundant and are called *spare* processing elements. The functional array contains interconnected PEs in which each PE performs the function of a cell in the logical array.

Definition 2.10 A *logical array* is a desired architecture structure which is a functional representation of a problem to be solved. □

Definition 2.11 A *physical array* is a hardware computing structure onto which a logical array is mapped. □

If we have a physical array of size $M \times N$, we can construct functional arrays of size $K \times L$, where $K \leq M$ and $L \leq N$. The target array of size $K \times L$ is considered as the logical array, while the physical array is of size $M \times N$. In Figure 2.2 $M = 5$, $N = 5$, $K = 4$ and $L = 4$.

To distinguish between locations on the physical array and the logical array, we call a location on the logical array a *cell* while a location on the physical array is called an *element* (see Figure 2.2). A physical array may contain faulty PEs. The location of faulty PEs greatly influences the outcome of many reconfiguration techniques.

Definition 2.12 The abstraction of the pattern (distribution) of faulty PEs in a processor array is called *fault pattern* of the processor array.□

The dashed line shown in Figure 2.2 shows a fault pattern.

Definition 2.13 *Spare demand* is defined as the ratio of total number of faulty elements (F) to the total number of redundant spare elements (S) provided in an array i.e. $\rho = F/S$. □

Spare demand in Figure 2.2 is 9/9.

An important figure of merit which relates to the yield of processor arrays is *survivability.* It is defined as follows.

Definition 2.14 *Survivability* is the probability with which a faulty array can be configured, when spare demand is given. □

The reconfiguration techniques for yield enhancement of processor arrays can be categorized into four classes. They are

- local reconfiguration,
- global reconfiguration,
- set switched (hierarchical) reconfiguration, and

- hybrid reconfiguration.

In local reconfiguration techniques, faulty cells are directly replaced with spare cells. Therefore, for a group of cells in the array there should exist a spare which can directly replace a failed module as shown in Figure 2.3(a). Interstitial redundancy [11] is an approach often referred to in the literature as an example for local reconfiguration technique. In interstitial redundancy, two or four cells share a common spare cell. The spare can be used for replacing any one of the adjacent cells. Therefore, inter-processor delay is fixed and less, but, percentage redundancy is high [11]. This technique also suffers from low survivability.

In global reconfiguration techniques a set of spares are commonly used for reconfiguration of faulty elements in the whole array (see Figure 2.3(b)). Generally, there is a spare row, spare column or both in an array. If an element is faulty then a sequence of replacements are made from the faulty to a spare element. In some global routing techniques no distinction is made between spare and non-spare cells. Initially, an attempt is made to construct the array by eliminating the faulty cells in the reconfiguration process. Unused non-faulty cells are termed as spares (can take any location in the array) and reconfiguration is performed to construct the final array. One such technique is the rule based reconfiguration technique [4].

Set switched or hierarchical redundancy combines the local and global reconfiguration techniques. The array contains equal sized subarrays. There is redundancy within a subarray and there are spare subarrays. The subarrays are individually

reconfigured first using local reconfiguration. Then the complete array is formed from those subarrays which have been reconfigured successfully. The two level redundancy approach [15] adopts two levels of redundancy, within a subarray and redundant subarrays. Figure 2.3(c) shows two levels of redundancy. Level 1 has local redundancy within a block and level 2 uses switches to reconfigure blocks.

The hybrid approach is a slight deviation from the hierarchical approach with global redundancy in both levels. Instead of using dedicated spares in each block, the spares are shared between neighboring blocks as shown in Figure 2.3(d). Since the spare elements are shared, we cannot switch blocks as in hierarchical reconfiguration approach. Therefore, spare blocks are not used in this approach [10].

Different types of switch fabrics are used for reconfiguration in global reconfiguration of processor arrays. Some of these switch fabrics are 3×3, 3×2, 2×3 and 2×2 switch bus interconnections. In switch bus architecture the switching resources are divided into two planes. Data flow from left to right in the array uses the horizontal plane for routing its signals, and the data that flows from top to bottom uses the vertical plane for routing of signals. The horizontal and the vertical planes are independent and there is no connection between the two planes. In Figure 2.4 the bold line interconnections form the horizontal plane and the dashed line interconnections forms the vertical plane. The 3×3 and 3×2 switch bus interconnections are defined below.

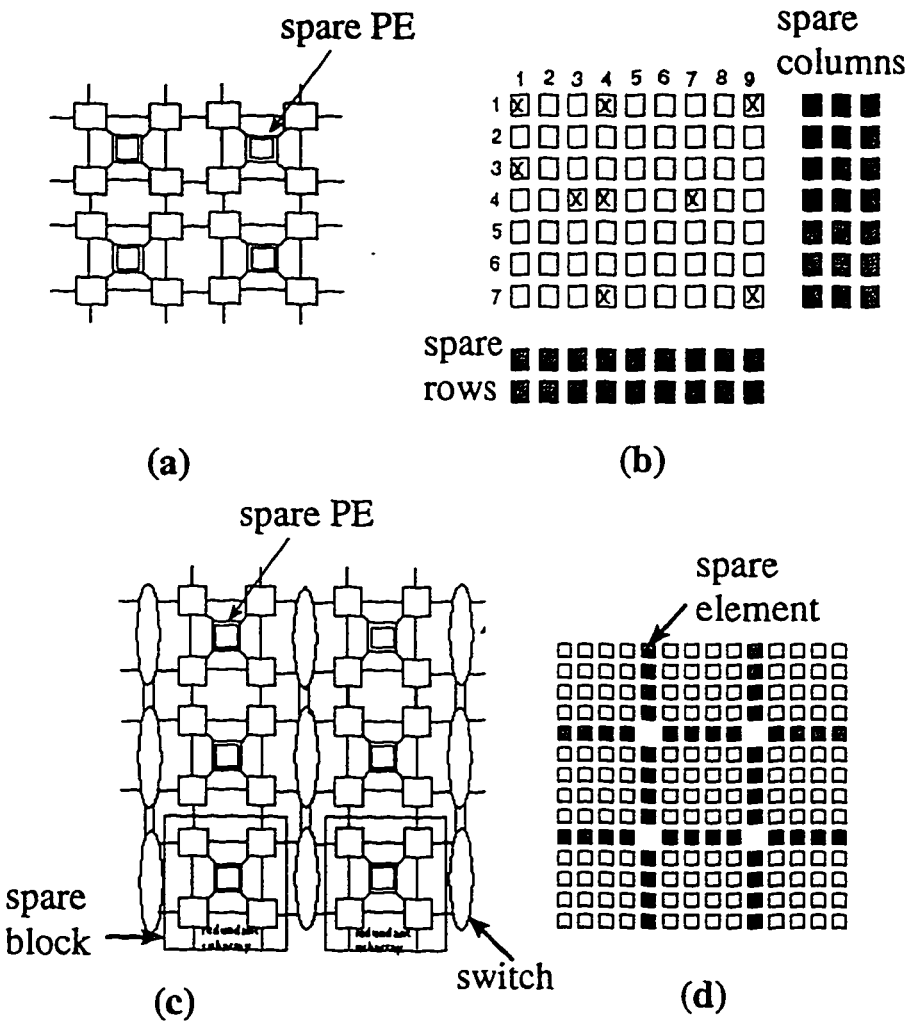


Figure 2.3: Different reconfiguration approaches (a) local redundancy (b) global redundancy (c) hierarchical redundancy and (d) hybrid redundancy.

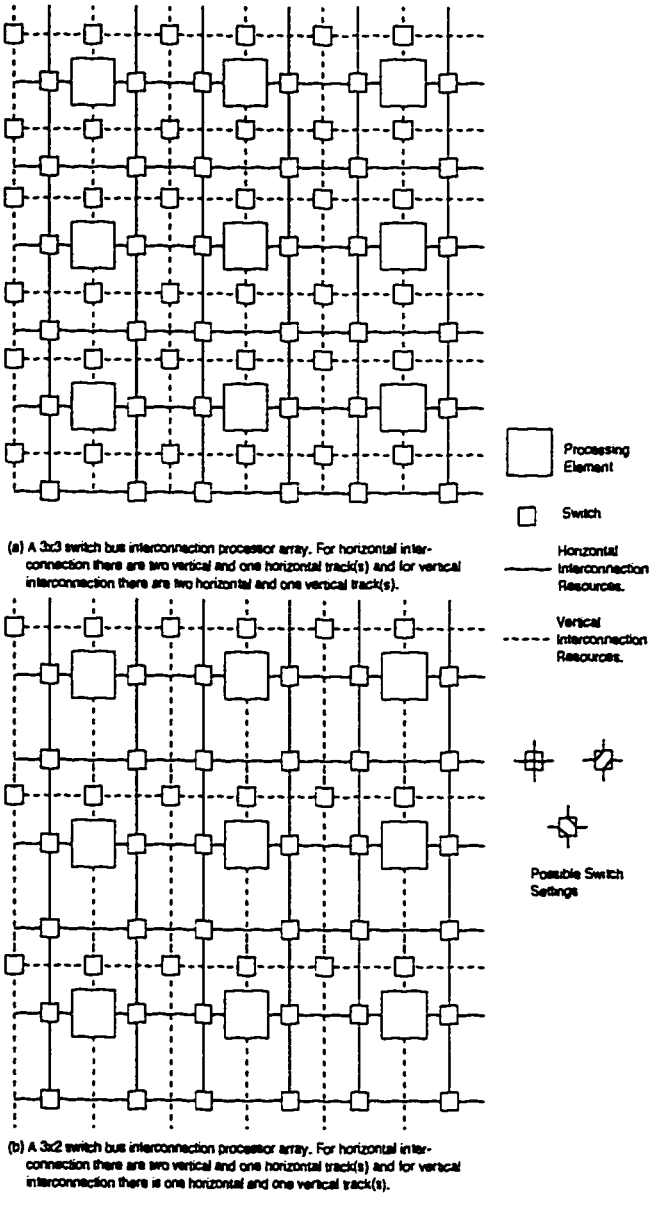


Figure 2.4: The switch bus interconnection network.

Definition 2.15 A 3×3 *switch bus* interconnection is defined as a switching fabric which contains two vertical tracks in each channel between columns and one horizontal track in each channel between rows for the horizontal plane. In the vertical plane it contains two horizontal tracks in each channel between rows and one vertical track in each channel between columns. □

Figure 2.4(a) shows a 3×3 switch bus.

Definition 2.16 A 3×2 *switch bus* interconnection is defined as a switching fabric which contains two vertical tracks in each channel between columns and one horizontal track in each channel between rows for the horizontal plane. In the vertical plane it contains one horizontal track in each channel between rows and one vertical track in each channel between columns. □

Figure 2.4(b) shows a 3×2 switch bus.

The switch bus interconnection is used in the rule based approach. In rule based global reconfiguration the array is divided into overlapping regions. The placement of each logical cell is restricted to a region. This region is called the domain of the logical cell.

Definition 2.17 The *domain* of a logical cell is defined as a set of physical elements in the physical array to which a logical cell can be assigned. □

To form these domains rule-based approach [21] uses the variable domain approach. In variable domain approach the array is divided into row and column ranges.

Definition 2.18 The *row range* of a logical row is defined as a set of physical elements in the physical array to which a logical row can be assigned. □

In variable domain approach the logical row i of a logical array of size $L \times K$ can be assigned to physical rows which contain $(i-1) * L$ non-faulty element to the physical row which contains $i * L$ non-faulty element when the array is scanned row-wise. For example in Figure 2.5(b) row range of logical row 4 starts from the row which contains non-faulty element $(4-1) * 4$ to the non-faulty element $(4) * 4$ (i.e. from physical row 3 to row 4.)

Definition 2.19 The *column range* of a logical column is defined as a set of physical elements in the physical array to which a logical column can be assigned. □

In Figure 2.5(a) column range of logical column 4 starts from the column which contains non-faulty element $(4-1) * 4$ to the non-faulty element $(4) * 4$ i.e. from physical column 3 to column 4.

The intersection of row range i with the column range j defines the domain for the logical cell $(i,j)_L$. For example in Figure 2.5(c) intersection of row range 4 and column range 4 defines the domain of logical cell $(4,4)_L$. Domain of logical cell $(4,4)_L$ contains physical elements $(3,4)_P$, $(3,5)_P$, $(4,4)_P$, and $(4,5)_P$.

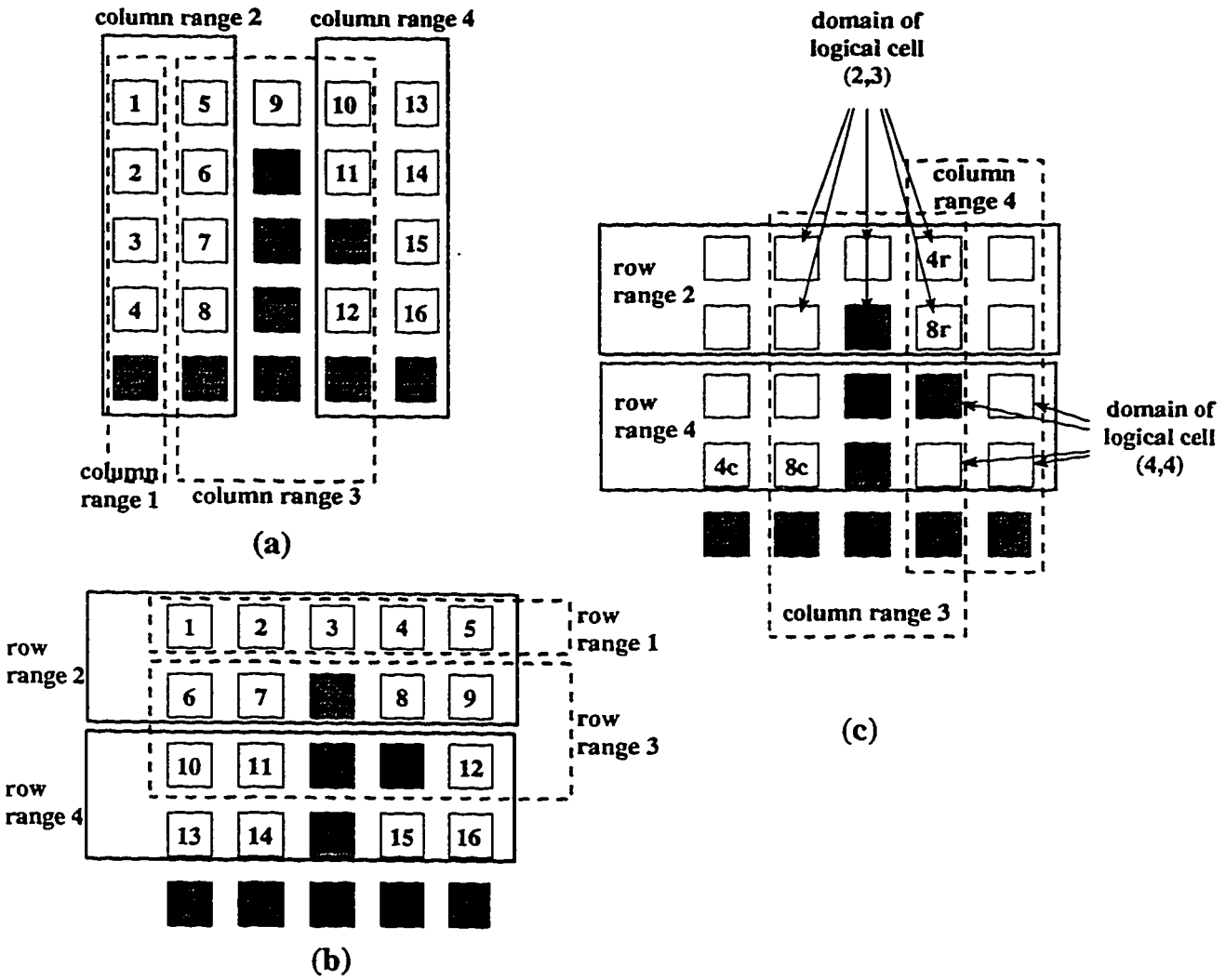


Figure 2.5: The variable domain approach. (a) Formation of column ranges. (b) Formation of row ranges. (c) Domains of logical cells $(2,3)_L$ and $(4,4)_L$.

Definition 2.20 The *Domain level 1* (1st domain, safe domain) of a logical cell $[i,j]_L$ contains all the physical elements in the overlapped region of row range i the columns range j.□

2.3 Background to the Proposed Framework

The framework for yield enhancement of processor arrays is a three layer implementation of the placement of logical cells to physical elements on the processor array. We begin the reconfiguration on the processor array from an initial solution.

Definition 2.21 The initial arrangement of logical cells on the physical array is called *initial solution.* □

The initial placement on the faulty array given in Figure 2.2 is shown in Figure 2.6. Some definitions related to the initial solution are given below.

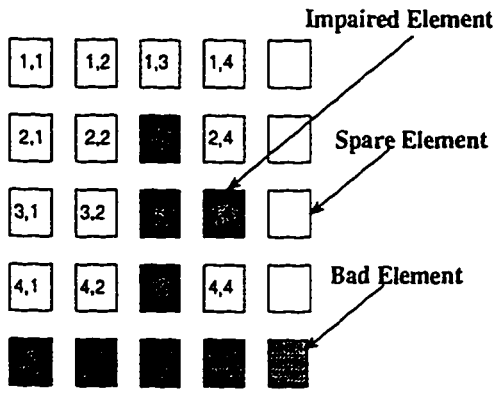


Figure 2.6: A possible initial placement of the logical array on the physical array (initial solution).

Definition 2.22 An element is *impaired* if it is faulty and a logical cell is assigned to it. □

Example of an impaired element is shown in Figure 2.6.

Definition 2.23 An element is *bad* if it is faulty and no logical cell is assigned to it. □

Example of a bad element is shown in Figure 2.6.

Definition 2.24 An element is *spare* if it is non-faulty and no logical cell is assigned to it. □

Example of a spare element is shown in Figure 2.6.

After obtaining an initial solution we have to make use of the three layers of the framework to reconfigure the faulty array. The first layer is the core of the framework and provides a method to avoid congestion in channels by using the rules for 3×3 and 3×2 switch bus interconnection (see Appendix A). If a placement of logical cells is expected to cause congestion then some of the rules in the core of the framework are violated. Reconfiguration mechanism can make use of this knowledge. Thus, in the first layer, the framework provides a mechanism to reconfigure through congestion avoidance.

In the second layer of the framework we provide utilities which aid the substitution of an impaired element with a spare element. The basic step in the process of substitution is *reassignment.*

Definition 2.25 When a logical cell (location) is taken away from an element and assigned to another element then the logical cell is said to have been *reassigned.* □

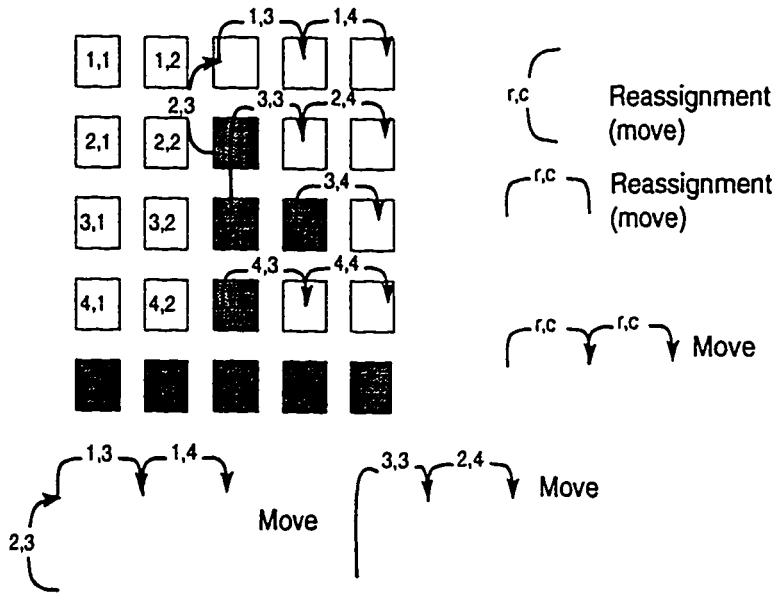


Figure 2.7: The reconfiguration process.

In Figure 2.7 each curved arrow indicates a reassignment. Reassignment of logical cell can be made from a PE to an immediate neighbor. In Figure 2.7 assignment of logical cell $(3,3)_L$ from element $(3,3)_P$ to $(2,4)_P$ is such a reassignment. To substitute an impaired element with a spare element we may have to make a chain of reassignments.

Definition 2.26 A chain of reassignments of logical cells from a source physical element (spare) to the destination physical element (impaired) is reassigned is called a *move* from the spare location to the faulty location. □

Some moves made to reconfigure the faulty array is shown in Figure 2.7.

The second layer of the proposed framework provides mechanisms which check all possible moves that can be made to substitute an impaired element with a spare element. Reassignment of an impaired element to a spare element causes reorganization of the placement of the logical cells. As more impaired elements are substituted (moved) with spare elements it becomes difficult to reconfigure the remaining impaired elements due to saturation of reconfiguration resources.

The third layer of the framework deals with the order in which the moves should be made. The sequence in which we make the moves is important as a few inefficient moves can exhaust the reconfiguration resources of the processor array.

Definition 2.27 A *sequence of moves* is defined as the sequence of moves of pairs of spare-impaired elements taken. □

When a sequence of moves is made the different intermediate placement of logical cells on the array will be different valid placements of the fault free array.

Definition 2.28 A physical array is a **fault free array** (FFA) if all its processing elements are non-faulty. □

The different valid placements form part of all possible placements on the fault free array. The complete set of these placements forms the sample space.

Definition 2.29 The *sample space of the FFA* contains all possible ways a given logical array can be mapped to a physical array. □

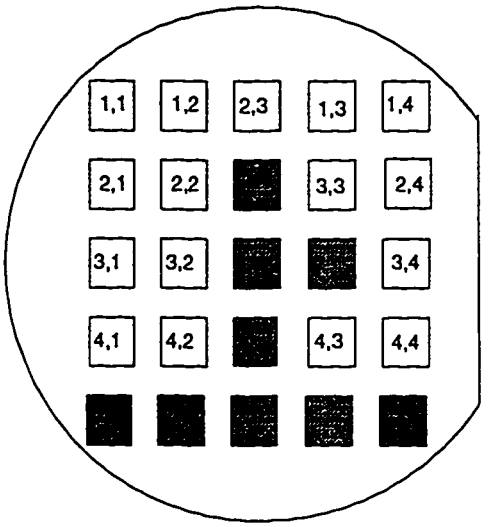


Figure 2.8: Final solution obtained after moves are made from the initial solution.

When the processor array contains faulty PEs then some of the placements are not valid. Therefore, the sample space of a faulty array is a subset of the sample space of the FFA.

Definition 2.30 The *sample space of a FA* is defined as the total number of ways in which the logical cells can be mapped to the non-faulty elements on the physical array. □

Using the knowledge of the sample space efficient algorithms can be developed to make the sequence of moves. The final configuration obtained after making the sequence of moves shown in Figure 2.7 is shown in Figure 2.8.

2.4 Concluding Remarks

In this chapter we have reviewed the basic concepts related to defect distribution and reconfiguration schemes in processor arrays. Defects have to be modeled in processor arrays. The two factors which play an important role in modeling defects are the the defect density D_i and clustering of defects λ_i. Using the fault models different fault patterns in processor arrays can be obtained. Survivability defines a measure of success for reconfiguring fault patterns obtained from defect models using different reconfiguration schemes. Different reconfiguration schemes were reviewed and the underlying switch fabrics are shown. The background to the proposed framework is explained with examples.

Chapter 3

Fault Distribution

Earlier VLSI circuits contained a small number of devices and consequently an acceptably low average number of defects per chip. Subsequent increase in circuit complexity resulted mainly from higher integration densities produced by reducing the device feature size. As feature size of transistors has been reduced to submicron level and increased circuit density is reaching the physical limits imposed by the minimum feature size of transistors for proper functioning, designers are expected to turn to large area circuits, full-wafer scale integration and multi-chip module technology [24].

Another consequence of increasing the number of devices on a VLSI chip is that the probability of having device failures (due to manufacturing and operational faults) also increases. Device defects are defined as any physical anomaly which causes a circuit fault. Some of these are shorts or resistive paths or opens caused by particles,

misalignment, photoresist splatters and flakes, weak spots in insulators, pinholes, opens due to step coverage problems, scratches, contamination, localized metallurgical anomalies, and crystallographic defects [1]. As the device failures increase, the number of good chips manufactured (yield) decreases.

A cost effective technique for improving the yield is to incorporate redundancy. The basic idea is to provide the system with extra (redundant) resources, beyond the minimum needed to achieve the system requirement. These extra resources can be used to avoid the use of faulty parts by restructuring the system (reconfiguration) to isolate the defective parts [5] [25].

An emerging technology for restructuring manufactured VLSI wafers is restructurable very large scale integration (RVLSI). In this technology a laser is used to restructure wafer scale circuits for customization and defect avoidance. RVLSI technology uses a laser-programmed antifuse, or link to make a connection between two metal lines, and laser melting of metal to segment a track into separate nets. It is reported that laser linking yield is greater than 99.9 percent [16].

RVLSI technology for reconfiguration has very high yield of interconnection i.e. almost non-faulty switching mechanism. However, yield of semiconductor integrated circuits is very low due to various manufacture defects. The achievement of a competitive or acceptable level of yield can be the primary factor in the success or failure of a particular product [26].

The rest of the chapter is organized as follows. In Section 3.1 we present an overview

of manufacture defect modeling. Section 3.2 talks in more detail about different types of defects and their effect on yield. In section 3.3 we present different models for fault distribution. In section 3.4 we present the outline of the fault distribution generator used in this thesis. In section 3.5 we present some concluding remarks.

3.1 Modeling of Manufacturing Defects

Manufacturing defects can be classified as *gross defects* and *spot defects*. Gross defects are large area defects which cause a complete wafer or part of a wafer to become non-functioning [5]. Spot defects are random defects and are five times more in number than the gross defects. While wafers with spot defects can be made usable using redundancy, it is very difficult to salvage gross defects. In this study, an emphasis is made on defect modeling for spot defects only.

An accurate yield model for predicting the average die yield is of extreme importance. Companies like IBM use yield models for several purposes [26] [27] [28]. Some of these are

- For a line producing many products, actual yields are compared to a model. This tells us what to work on to improve yields of those products with yields below expectation.

- Yield models are used to predict manufacturing costs for products under development by determining the amount of chips that have to be manufactured

to get the desired amount of good chips.

- Yield models are used to define the maximum level of integration possible for a given VLSI implementation.

Yield models are, generally a function of D_0 the average number of defects per unit area, and A the chip area.

The first model used to predict IC yields was derived from the poisson probability distribution function. The poisson distribution makes a random distribution of defects. Thus causing scattered defects on the circuit. As the chip size continued to escalate, the poisson model became increasingly inaccurate and tended to underestimate the yields for large dies [26].

A yield model which gives similar results to those obtained using the poisson distribution is the binomial distribution. However, like poisson distribution this distribution is also inaccurate. One of the main reason that leads to the inaccuracy is clustering of manufacturing defects during chip fabrication [29].

Murphy reasoned that compounding the binomial distribution with a normalized distribution function gives a more accurate estimation of yield [26]. The most popular compounding function is the gamma function [30]. Compounding the binomial function with gamma function results in the negative binomial function. The negative binomial distribution has two parameters: the average number of defects λ and

a clustering parameter α. Negative binomial distribution for n defects in an area size A is given by [30]

$$\text{Prob}\{\text{there are } n \text{ defects in Area } A\} = \frac{\Gamma(\alpha+n)}{n!\Gamma(\alpha)} \frac{(\lambda/\alpha)^n}{(1+\lambda/\alpha)^{\alpha+n}}$$

where $\lambda = \lambda_1 + \lambda_2 + \cdots + \lambda_n$ and the indices indicate different defect types. Parameter α determines the extent of defect clustering on the circuits. Having very low values of α (approaching 0) causes all the defects to concentrate in a small area, while large values of α (greater than 5) makes the distribution approximate a poisson distribution. The clustering paramter has to be chosen carefully to give a good fit of the actual manufacturing defects. Therefore, depending on the size of clusters of defects, defect clusters in integrated circuits have been divided into four categories. They are small-size clustering, large-size clustering, medium-size clustering and clusters that vary in dimension. Depending on the size of a cluster, more accurate models have been proposed for the first three categories [31] [8] [32]. Efforts made to model the fourth class are still not definitive [5].

Recently, a more accurate model called the small and large scale model has been proposed. The author states that combination of the small-size model and the large-size model gives the best fit [33].

Different defect models give statistical distribution of defects, but do not give any idea about the spatial distribution of defects. Therefore, we have to simulate a spa-

tial distribution on wafer such that it exactly fits one of the models. A program to get a spatial distribution of defects is given in [3]. We have implemented a defect pattern generator based on the large-size model for defect distribution in processor arrays on the grounds proposed in [3].

3.2 Effect of Defects on Yield

In the manufacture of semiconductor integrated circuits, millions of electronic devices are produced simultaneously in a series of very complex processing steps. The VLSI chips themselves typically contain millions of such devices. The chances that all the devices and their interconnections will function according to the design depends on the control exercised in their manufacture.

All the manufacturing anomalies effect the proper functioning of a chip. When some of the manufacturing parameters are not set properly then they result in many defective chips on wafers. To estimate variations in the number of defective chips manufactured in a lot with the variation of different manufacturing parameters, the most widely used performance measure is *yield*. The fraction of chips that, upon completion of manufacture, can meet the set of very stringent final test requirements determines the *yield* of the manufacture line [1].

The yield associated with integrated circuit manufacturing can be divided into three

parts. The first part contains those chips which never reach final test. This is caused by the breakage of wafers, or it may be the result of process errors. Such types of losses are generally not treated in yield analysis. The second type of yield is the "final test yield". This is the yield that takes the localized process defects into account. The third part of integrated circuit yield has to do with packaging [1]. The packaging yield depends on the accuracy of the automated wire bonding equipment. The packaging yield is not treated in this work as we are mainly concerned with the final test yield.

The final test yield can be categorized into two classes. One class contains "gross defects" which have a "gross yield" associated with them. The second class of final test yield contains the "random defects" and is known as the random defect yield [1].

Gross defects are usually caused by manufacturing errors that cause parts of a wafer or a entire wafer to have no functioning chips. Excessive variations in process timing of "hot process" steps like diffusions, ion implants, epitaxial silicon growth or insulation deposition processes, cause failure of an entire wafer. Scratches from wafer mishandling, mask misalignment, and over and under etching cause large area defects. Gross yield is also caused by error in the photolithographic process. The resulting failures will cause gross chips failure on the wafer, making the complete wafer or some parts of the wafer unusable (see Figure 3.1 [1]). This is why we call them gross failures, and their associated yield, a gross yield [1]. Gross yield can

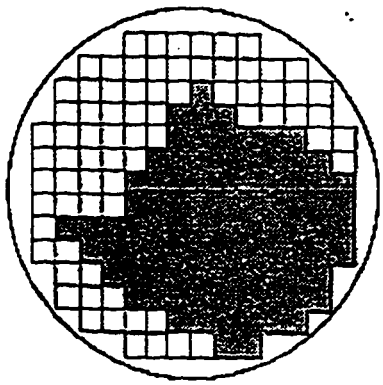


Figure 3.1: Gross yields affect whole wafers or parts of wafers. The black chips are functioning; the white chips are defective. (Scanned from [1])

be controlled by setting the manufacturing parameters. Therefore, well established manufacturer lines have been able to attain low loss due to gross yield.

Random defects are defects caused by different manufacturing processes, that cause defects to be formed randomly on the wafers. Random defects are generally caused by sources that are not controlled by manufacturing parameters directly. Elimination of these defects even if possible is not cost effective. Therefore, losses due to these defects are considered after fabrication. One approach is to use redundancy in the form of spare modules and interconnection mechanism for reconfiguration to substitute faulty modules with spare modules. However, the percentage of defective chips that can be made usable by the elimination of defective modules is affected by the presence of clusters of defective modules. Studies conducted to determine the location of defects have shown that random defects tend to cluster

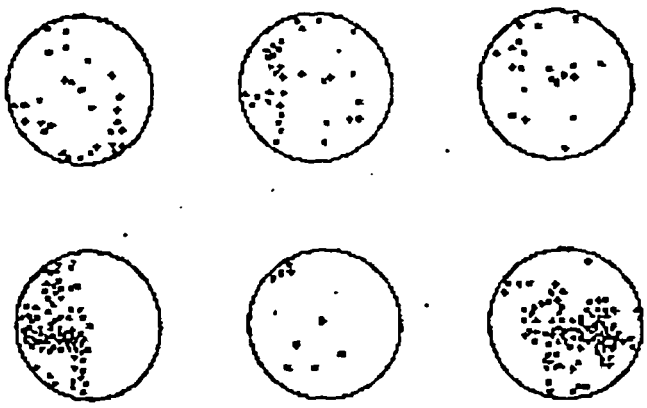


Figure 3.2: Particle locations on "dirty" wafers show a tendency towards clustering. (Scanned from [1])

[1] (See Figure 3.2, scanned from [1]).

Random defect types are pinhole defects, photo defects, leakage defects, etc. Pinhole defects and photo defects contribute to a major portion of yield loss.

Dielectric pinholes are microscopic defects in insulators - like silicon dioxide, silicon nitride, quartz and polymide - which are used between the conductive layers of the integrated circuit chips. These defects will cause short circuits between the conductors on different levels, thus causing chip failure. The area sensitive to pinholes is the area where conductors cross each other. The sum of all such overlap areas on a chip is called the *pinhole critical area* for that chip.

Random photo defects are caused by minute particles. These particles are deposited on wafers in various process steps. They are carried in the gases, vapors, solvents, photoresists, and other chemicals that are used to manufacture integrated circuits. Particles on wafers are clustered. A cause of clustering of particles is aggregates

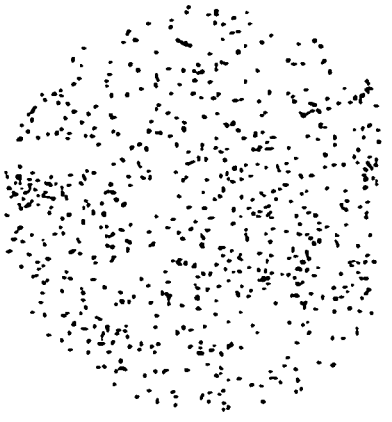


Figure 3.3: Particle locations measured by laser-based particle detector in controlled environment. (Scanned from [2]).

of particles that are collected in the manufacturing machinery. When shaken loose by vibrations, pressure changes, or gas flow changes, these clumps of particles will form clouds in the gases or liquids used for processing. Where, such clouds reach the wafer surface, particles will be clustered. Figure 3.3 (Scanned from [2]) shows a particle locations measured by laser based particle detector .

Excessive reverse-biased current in semiconductor diodes is another failure that is usually caused by defects in the silicon crystal lattice. This phenomenon is known as *junction leakage*. These leakage defects have also been found to cluster.

Gross yield loss accounts to 16.5%, in contrast to 83.5% of yield loss due to random defects, as shown in Figure 3.4 [1]. More recent work reports gross yield loss of 4.26% only [6] [7]. The random defects, therefore, cause atleast five times more chips to fail than those caused by gross defects.

Defect modelling for random defects have received more attention as the percent-

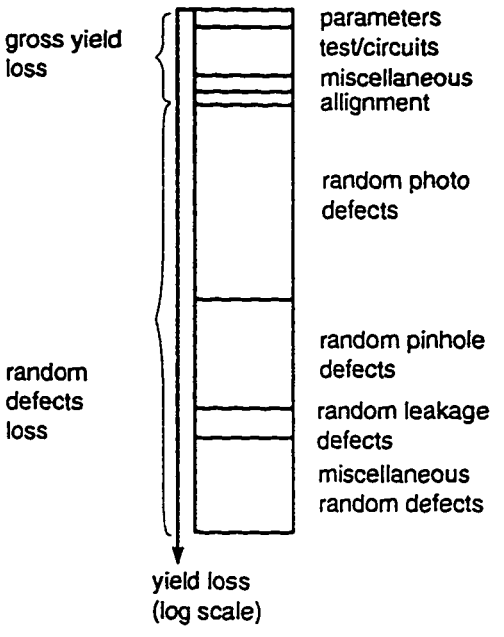


Figure 3.4: Relative yield loss on a logarithmic scale. Random defects cause most of the defects.

age of yield loss due to random defects is high. In this work we will consider defect modelling for random defects only.

3.3 Defect Modelling

A chip has to pass through a sequence of steps until it reaches the final testing stage. Each step may introduce defects in the chip. At the testing stage, faults in the chip can be considered as the cumulation of defects introduced by each step.

Therefore, the model for calculating average number of failures has the form [1]

$$\lambda_i = A_i D_i$$

where, λ_i is the average number of failures or defects caused by a defect type indicated by the indices i. The quantity A_i is the critical area. The defect density is designated by D_i and has the units of defects per unit area. Each defect mechanism has its own defect density. Quartz pinholes, for instance, are different from the thin-oxide pinholes, and they will therefore, have different defect densities. The overall average number of defects λ is the summation of defect averages during all steps [1].

$$\lambda = \sum_{i=1}^{m} \lambda_i.$$

A defect distribution model which was used earlier is the *Poisson model.* This is given by the equation [30]

$$P(i) = e^{-D_0 A_{chip}} \frac{[D_0 A_{chip}]^i}{i!} = e^{-\lambda} \frac{\lambda^i}{i!}$$

where, $P(i)$ is the probability of finding i defects on a chip, D_0 is the average defect density and A_{chip} is the area of the chip. A defect distribution on a wafer using this model is shown in Figure 3.5 (scanned from [2]). This formula is considered too pessimistic for present day VLSI/WSI technology, as it does not take clustering into account. Therefore, other models were proposed which take the effect of clustering into consideration [2] [1] [30] [34] [35] [8] [31] [32] [33]. A good defect model for

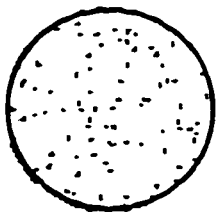


Figure 3.5: Simulated defect map showing random or poisson defects. (Scanned from [2]).

integrated circuit chips has to take two important factors into account. First, is the variation of wafer to wafer defect densities and the second is defect clustering. A good way of integrating the effect of these factors is to use compound Poisson statistics [30]. The Poisson distribution is compounded with a function $f(D)$, which represents the normalized distribution of chip defect densities

$$P(i, D_0, A) = \int_0^\infty \frac{(D_0 A)^i e^{-DA}}{i!} f(D) d(D).$$

This function is also known as murphy's formula. The function $f(D)$ is, in effect, a weighting function which accounts for the non-random distribution of defects. Different functions have been proposed and used for $f(D)$. Some of these functions are delta function, triangular function, exponential function, rectangular function and gamma distribution function [30]. The most widely used function is the gamma distribution function. Gamma distribution has the form

$$f(x) = \frac{\beta^{-\alpha} x^{\alpha-1} e^{-x/\beta}}{\Gamma(\alpha)} \qquad if \quad x > 0$$

where, α and β are parameters of the probability distribution function. The parameters should be defined correctly based on their physical interpretation [36]. The mean of this distribution is $\alpha\beta$ and variance is $\alpha\beta^2$.

For defect distribution the gamma distribution has the form

$$f(D) = \frac{1}{\Gamma(\alpha)\beta^{\alpha}} D^{\alpha-1} e^{-D/\beta}$$

where, $\Gamma(\alpha)$ is the gamma function, α and β are given by

$$\alpha = \frac{D_0^2}{var(D)}, \qquad \beta = \frac{var(D)}{D_0}, \qquad D_0 = \alpha\beta.$$

where $var(D)$ is the variance of defect density.

For defect density distribution, we require only one parameter, i.e. clustering parameter. Therefore, we eliminate parameter β. Substituting the resulting function into murphy's formulae gives

$$P(i, \alpha, D_0, A) = \frac{\Gamma(i+\alpha)(AD_0/\alpha)^i}{(i!)\Gamma(\alpha)(1+AD_0/\alpha)^{i+\alpha}}.$$

where α is the clustering parameter.

Substituting $\lambda = AD_0$, we get the negative binomial distribution function

$$P(X = i) = \frac{\Gamma(\alpha+i)}{i!\Gamma(\alpha)} \frac{(\bar{\lambda}/\alpha)^i}{(1+\bar{\lambda}/\alpha)^{\alpha+i}} \tag{3.1}$$

where, $\bar{\lambda}$, is the grand average of λ_is of all areas on the chip. A spatial distribution of the negative binomial equation obtained through simulation is shown in Figure 3.6 (scanned from [3]).

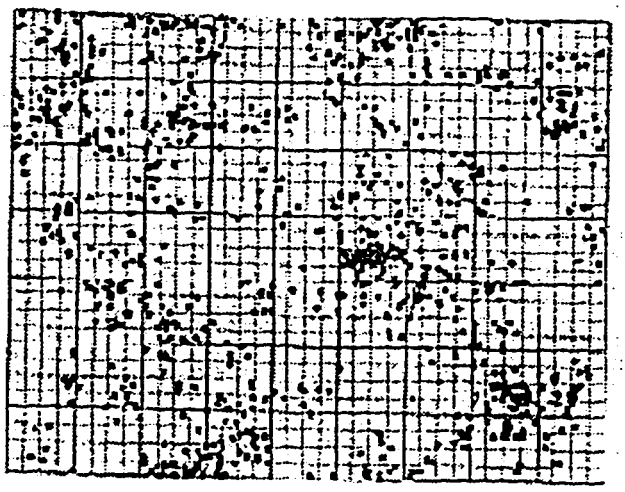


Figure 3.6: A map generated with a negative binomial cluster generator. (Scanned from [3]).

3.3.1 Variations in Clustering

Most of the clustering is expected to be caused by wafer-to-wafer variations of defect densities [8]. Another source of clustering is the radial variation in the average number of defects per chip [37].

In order to understand the distribution of defects, Meyer and Pardhan [38] divided the wafer into sections known as *quadrants* (also called *windows*). The size of the quadrants can be varied to correspond to various size products. If a quadrant is small, it will tend to be contained within a cluster of defects and therefore will not reflect the extent of clustering. As the quadrant size is increased, it is more likely that an entire cluster would fit neatly within a quadrant, so the clustering observation is maximized, yielding a distribution fit with a lower α. As the quadrant size

is increased further, entire clusters may fall within a quadrant, together with much area free of defects, so the clustering is less apparent and α starts to rise.

As $\alpha \rightarrow \infty$ represents random defects and complete absence of clustering. When $\alpha = 0$, the defects are clustered in infinitely small regions and none are found elsewhere. Actually, values of α range between 0.3 and 5 [8] [26].

3.3.2 Defect Modelling based on Clustering

Clusters that occur on VLSI/WSI chips can be categorized into three classes. The first pertains to clusters that are much larger than the chip size (*large size clustering*). The second class deals with defect clusters that are smaller than chip area (*small size clustering*). The third class contains clusters with the same dimension as that of the chip area (*medium size clustering*).

In *large-size clustering* the size of a cluster is comparable to the size of the wafer, implying that the number of defects in any subarea of the wafer has a negative-binomial distribution, and that the parameter α is constant for any subarea of the wafer [8]. In *small-size clustering* the wafer is divided into small modules (considered unit size) that are statistically independent. Any subarea of the wafer is assumed to consist of a whole number of these modules, and hence defects in any disjoint subareas are independent. In addition, the number of defects in any subarea has a negative binomial distribution, and the parameter α is proportional to the area

[31]. *Medium-size clustering* is an intermediate method, in which each module is considered in its entirety, and the different modules are then combined relying on their statistical independence to form blocks. According to this distribution the basic unit is a block consisting R rows and C columns of $R \times C$ modules, the number of defects in a block has a negative binomial distribution, and the defects in distinct blocks are statistically independent [33]. The size of the block has to be chosen such that it encompasses a cluster.

A *unified approach* to the three distribution schemes is presented in the work by Koren et. al. [32]. In this model a chip consists of basic units called modules. A module is a circuit block such as a memory subarray or a digital logic macro that is replicated in a chip. The area of a module is assumed to be the unit area, and all other areas are measured in these units. Therefore, a block is composed of B modules and the wafer of W modules, respectively. There are W/B blocks on the wafer. It is assumed that the number of defects per wafer has a negative binomial distribution with parameters $(\lambda_w,\ \alpha_w)$.

The authors proved the following:

- The number of defects in a block has a negative binomial distribution with parameters $(\lambda_b,\ \alpha_b)$ where

$$\lambda_b = \frac{\lambda_w}{W/B};\quad \alpha_b = \frac{\alpha_w}{W/B}$$

- For any area of size A contained in one block, the number of defects has a negative binomial distribution with parameters

$$\left(\frac{A}{B}\lambda_b, \alpha_b\right).$$

Therefore, the module parameters (λ_m, α_m) are given by

$$\lambda_m = \frac{\lambda_b}{B} = \frac{\lambda_w}{W}; \quad \alpha_m = \alpha_b = \frac{\alpha_w}{W/B}.$$

- For any area consisting of C blocks, the number of defects has a negative binomial distribution with parameters

$$(C\lambda_b, C\alpha_b).$$

- Let E be the size of an area which is divided among K blocks such that

$$E = \sum_{i=1} KA_i$$

where, A_i is the size of the subarea contained in the ith block. The number of defects in area E has a negative binomial distribution if and only if $A_1 = A_2 = \cdots = A_K = E/K$; otherwise, it is the sum of K statistically independent negative binomial random variables.

A schematic showing modules, blocks and wafer is shown in Figure 3.7. In this figure, the size of the block is 2×3 modules, $B = 6$. The wafer contains 54 modules,

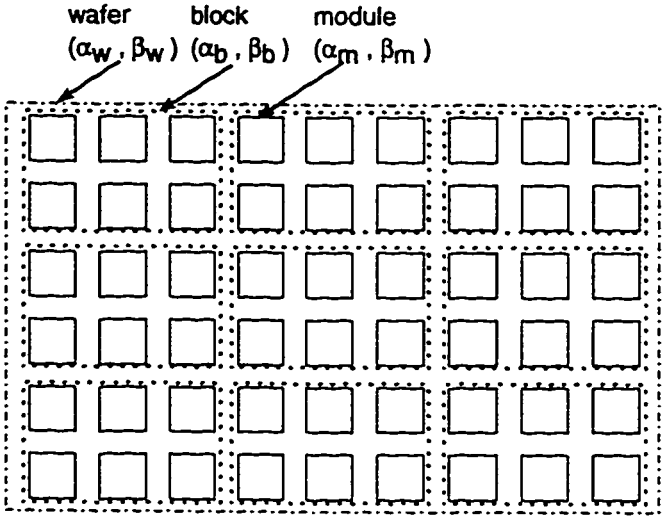


Figure 3.7: A schematic showing 3 × 3 blocks on a wafer. Each block contains 2 × 3 modules.

with 3 × 3 blocks and 6 × 9 modules. Value of parameter $W = 54$.

The correct estimation of block size, of λ_m, and of α_m is of crucial importance for the proper evaluation of VLSI chips.

As an extension to the unified approach, a combination of the small-size model and the large-size model has been proposed by Koren et. al. [33]. According to this study there are two independent defect sources. Koren et. al. combined different distributions to study the effect. The small-size model combined with the large size model produced the best fit to the empirical distribution. The other distribution which is comparable to this distribution is the medium size model.

3.4 Defect Distribution for Processor Arrays

Defects on wafers are distributed according to defect sources due to different manufacture anomalies. The average number of defects per unit area (module) depends on the contribution made by each defect source. Due to the very nature of the source of random defects, defects tend to cluster. Therefore, two factors decide the nature of defect distribution on wafers. These factors are defects per unit area (module) λ and the clustering parameter α.

Modelling defects distributions in processor arrays for WSI is similar to modelling defects on wafers. This is because the defect sources and the nature of defects remain the same.

To determine modelling parameters α and λ, the unit area (module area) has to be chosen carefully. All other areas are measured in terms of unit area. The unit area is defined as a module by Koren [32] [33]. It is suggested in [32] that the module should be a circuit block such as a memory subarray or a digital-logic macro that is replicated in a chip.

In the context of processor arrays, we can consider that PE corresponds to a module. Depending on the size of the PE and manufacturing anomalies of a manufacturing site, we have to use a model that gives an accurate fit of defect distribution. If the size of the PE is large, then defect cluster size may be bounded by the size of the PE. In this case we have to use small-size model [31]. Defects within a PE would then

be distributed according to negative binomial distribution, while the distribution of defects in each PE is a statistically independent function. When size of the PE is smaller than the cluster size then we should use the large-size model, medium-size model or combination of small-size model, large-size model and medium-size model [33].

The assumption that the basic module is a processing element will make the process of generating a defect distribution feasible. If we are given the size of the processing element, and defect density & clustering parameter of the manufacturing site, we can obtain statistics of defect distribution for a lot of chips manufactured, using the negative binomial equation 3.1 .

Using the negative binomial defect distribution we have to make a spatial distribution of defects to determine the location of faulty processing elements. Moreover, the fact that defects are generated as a function of time, has to be taken into consideration. A method to generate a spatial negative binomial distribution of defects which is a function of time is given in [3].

The negative binomial cluster generator [3] simulates the defect distribution using randomly generated defects as a function of time. The array is divided into equal sized modules. During an incremental time interval Δt, the probability of adding a defect to a module on a chip (wafer) is linearly related to the number of defects already on that module. The abstraction is implemented by generating a random number which is used to determine whether a defect is to be added to a chip during

a time interval Δt. The probability of adding a defect to a given chip is assumed to be linearly dependant not only on the number of defects present on that chip, but also on the number of defects on its four nearest neighbors. For example, if the probability of adding a defect to a chip is set to 0.8 then a defect is added only if the random number generated (between 0 and 1) for a module on the chip is greater than 0.8. When a defect is added then the probability of adding the defect is retained on the module of the chip (wafer). When generating another defect on the same module, the new generated random number is summed with a fraction (weight) of the retained value and a fraction of the retained values on its neighbors. Thus the probability of accepting another defect on the module is increased. This makes the cluster grow.

The negative binomial cluster generator can be used to generate defects based on large area clustering as well as small area clustering. Clustering parameter and defect density vary with the size of the window. Therefore, these factors have to tuned using the equations given in section 3.3.2. Medium area clustering can also be simulated, by distributing defects independently in each block. Small and large area clustering can be simulated by generating distribution of defects for small area clustering and large area clustering independently, and combining both distributions in the end.

Many distribution models that were developed earlier were developed for large area clusters [8] [1] [26] [38]. Due to the easy availability of modelling methods and param-

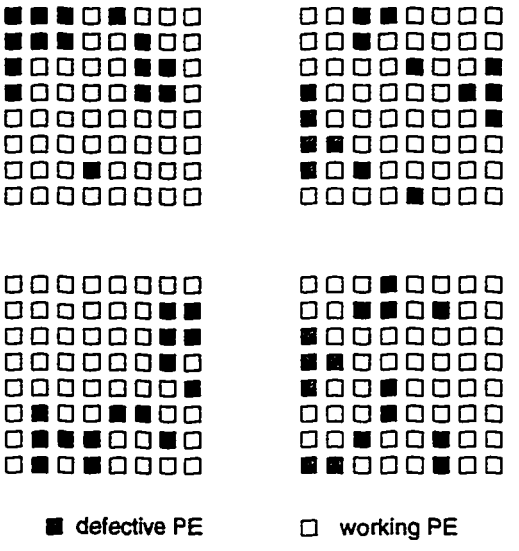


Figure 3.8: Fault patterns obtained for 8 × 8 array with 15 faulty PEs and $\alpha = 1$.

eter values based on statistical data obtained from manufacturing sites in large-area clustering, we have based our defect distribution generator on this model. Defects are generated on the chip (wafer) the same way as given in [3]. After obtaining the defect locations on a chip (wafer), the simulator determines the corresponding locations on the processing elements in the processor array. It is assumed that all defects cause faults. If the number of faulty processing elements is within the desired range then the fault pattern (distribution of faults) on the processor array is accepted. Some patterns for 8 × 8 array with 15 faulty PEs, $\alpha = 1$ are shown in Figure 3.8. Figure 3.9 shows some fault patterns in 8 × 8 array with 15 faulty PEs and $\alpha = 0.3$.

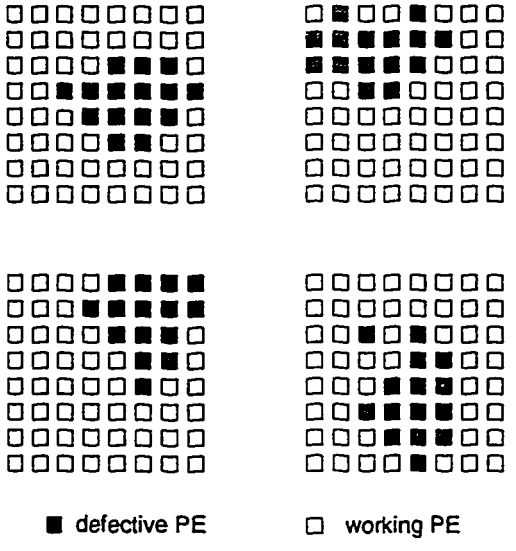


Figure 3.9: Fault patterns obtained for 8×8 array with 15 faulty PEs and $\alpha = 0.3$.

3.5 Concluding Remarks

As the size of the circuits became larger, IC manufacturers produced chips with smaller device size to reduce cost of manufacture. However, as the device size is approaching the physical limits of a transistor, alternate methods have to be sought to meet the ever increasing demand of larger sized circuits. WSI technology is currently receiving a great deal of attention. This trend is expected to continue in the foreseeable future. However, the main obstacles of having low yield has to be solved in order to meet the requirements of the market. Yield enhancement using interconnection resources for restructuring circuits is gaining importance. Though many architectures and techniques have been proposed for yield enhancement, realistic estimates using practical defect models are lacking. One of the reasons is

the unavailability of modelling parameters which exactly fits defect distribution in a commercial IC fabrication sites of processor arrays. Another reason is the unavailability of spatial defect distribution models which reflect fault patterns actually formed on ICs. It is very desirable to have benchmarks of fault patterns on ICs for different sizes of processor arrays with varying size of processing elements.

A common assumption made in processor arrays is that interconnection mechanism is fault free. Such an assumption is not valid if silicon based switches (soft switches) are used [39]. However, it is expected that faults in channels between processing elements have a much smaller failure rate if laser programmed antifuse based switches (hard switches) are used. Therefore, hard switches can be used for yield. Though hard switches occupy lesser area than soft switches, the cost in terms of chip area and the time it takes to restructure the array after manufacture is substantial [16]. Therefore, there is a need to minimize interconnection redundancy as it will reduce cost as well as reduce chances of failures in channels between PEs.

Chapter 4

Hardware Reconfiguration Techniques for Improving Yield

Due to complex structures in VLSI with more than 10^5 gates per chip the defect size that can make circuits faulty, becomes very small (μm) [18]. z Due to the presence of faults in ICs, yield at the time of manufacturing falls. Due to lesser yield, the manufacturers incur more cost to manufacture ICs.

The following steps have been suggested to improve the processing yields of VLSI circuits [18]:

1. modifying the design rules, thereby reducing the probability of yield loss due to the critical spacings, such as contact to poly layers (defect avoidance);

2. modifying the design methodologies, such that redundant elements can be introduced on the chip to compensate for defective areas (defect tolerance).

Improvement in yield due to defect avoidance is limited, therefore, yield improvement through introduction of redundancy is being used especially in memory circuits [24]. However, increasing amount of redundancy may prove counter productive as yield falls of sharply (approximately exponentially) with increasing chip area [19]. Depending on the assumptions made on the amount of redundancy used and the average number of wires in the intermodule interconnection area, the effect on the yield improvement will vary [18]. Many hardware redundancy techniques for different combinations of spare PE redundancy and interconnection redundancy have been studied for fault tolerance in processor arrays [15] [10] [11] [13] [4] [40] [41] [42].

Hardware reconfiguration schemes can be divided into four categories: local reconfiguration schemes, hierarchical reconfiguration schemes, global reconfiguration schemes and hybrid reconfiguration schemes. In local reconfiguration schemes the spare processing elements directly replace the faulty PEs. Therefore, more redundancy is required to provide spares for direct replacement. In global reconfiguration schemes spare PEs are global and the array has to be restructured to substitute a faulty element with a spare element. Hierarchical reconfiguration and hybrid reconfiguration schemes combine the local and global reconfiguration approaches. In these approaches the array is divided into smaller sized subarrays which have spare

elements available globally within the subarray. A comprehensive description of taxonomy of hardware-redundancy reconfiguration schemes is given in [9].

4.1 Local Reconfiguration Schemes

In local reconfiguration schemes, locality of processing elements is of prime concern [11] [12] [43] [44]. Therefore, these approaches rely on direct replacement of failed modules with spare modules or systematic reconfiguration strategy which preserves locality. It guarantees that the delay associated with interconnection paths after reconfiguration, will be within pre-defined limits. Therefore, such schemes are most suitable for synchronous array processors in which all the operations are synchronized according to a global clock. To preserve locality, the utilization of spares in the whole array is sacrificed. Therefore, yield improvement is expected to be limited.

4.1.1 Interstitial Redundancy Approach

In this technique, the objective of the reconfiguration strategy is to achieve fair area efficiency by using as many good PEs as possible, while at the same time ensuring shorter interconnection communication links. The spare PEs are assigned to failed primary cells in a prearranged clusters of two dimension arrays. For instance, in a 25% redundancy array as shown in Figure 4.1, every spare is assigned to a cluster

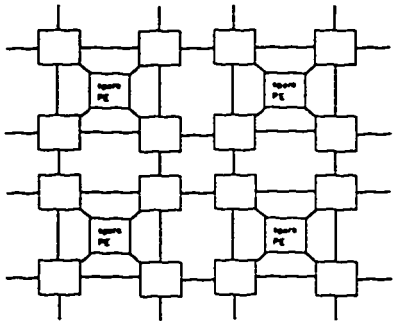


Figure 4.1: A (1,4) interstitial redundancy array.

of four primary cells [11].

The spare cell in the interstice of a subarray can be used for reconfiguration of the four cells in that subarray only. Therefore, interconnection delay after reconfiguration is increased slightly and does not exceed a pre-specified limit. The drawback is that failure of two cells in a subarray will lead to failure of reconfiguration of the whole subarray and consequentially the whole array.

Analytical yield estimation for $(1,4)$ interstitial redundancy array were carried out with the assumptions that all PEs have same survivability, switches are fault free, switches and interconnection occupy negligible area and layout area is directly proportional to total PE area. If the survival probability of each PE is p, then for an $n \times n$ target size array, the array yield is given by

$$y(1,4) = (p^5 + 5p^4(1-p))^{(n^2/4)}$$

For $p = 0.8$ and $n = 8$, survivability is 0.006723. For $p = 0.9$ and $n = 8$, survival probability is 0.256785. Therefore, with the assumptions made for analysis,

we obtain a very optimistic survivability which is very low even for higher values of survival probabilities ($p = 0.9$). Moreover, high survival probability of PE contradicts the assumption that size of the PE is large compared to the size of the switch as yield falls of exponentially with increasing chip area [19].

Other possible percentages of redundancy which can be introduced is 100% and 50% redundancy. But, it is has been reported that the yield improvement saturates above a limited amount (15%) of redundancy [18]. This indicates that chips should be designed around an optimal amount of additional logic on the chip to improve the yield [18]. Due to very high percentage of redundancy required in interstitial redundancy approach, improvement in yield may be limited.

4.2 Global Redundancy

In global reconfiguration techniques many algorithms can be mapped on the same interconnection structure. The main issue in this type of approach is utilization of spares. It is argued that the delays due to interconnection are not as crucial in some array types e.g. asynchronous array processors. Therefore, some global reconfiguration approaches relax the locality constraint [45] [46].

Before describing global routing approaches, we need to take a look at some of the available interconnection networks for processor arrays.

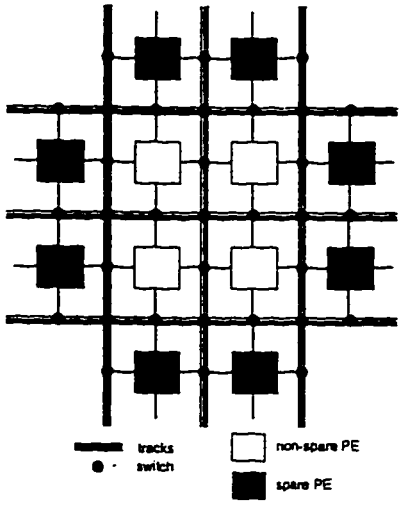


Figure 4.2: The 3-track-1-spare reconfiguration model.

Multiple Track Models: The 3-Track-1-Spare-Approach

This model consists of an $n \times k$ rectangular array with *one* row/column of spares on each side as shown in Figure 4.2. There are three tracks running along each grid line and the switches are located as shown in Figure 4.2. The architecture can be generalized to m-tracks problem. The connection between switch and PE is considered as $\frac{1}{2}$-track. Therefore, the generalized model is called $m\frac{1}{2}$-track problem [47] [48] [49] [23].

The faulty PE's need not act as switching elements. The main feature of this type of model is that it allows any given set of continuous and non-intersecting compensation paths between processing elements, if the switch fabric is non-faulty. This is shown through various examples in [50].. An algorithm for reconfiguring a faulty 3-track-1-spare array is given in [47].

Switched Bus Type Interconnection Network

In this model, switched bus is inserted in channels between each pair of rows and columns. In Figure 2.4 the small squares represent switches. The possible ways of setting the switches is also shown. The figure shows a 3×3 and 3×2 switch bus interconnection fault tolerant processor array (FTPA). The 2×2 model is shown in [21]. The interconnection resources are divided into row and column interconnection resources. Each track and associated switches are dedicated to the construction of a logical row or column, but not both. The column interconnection resources are used only for the construction of logical columns, and the row interconnection resources are used only for the construction of logical rows. In Figure 2.4 row interconnection resources consist of one horizontal track and two vertical tracks (1×2 column resources), and column interconnection resources consist of two horizontal tracks and one vertical track (2×1 row resources).

Many reconfiguration schemes use the switched bus type network due to its high regularity and simplicity which suits VLSI/WSI implementation. Many algorithms have been proposed for reconfiguration of the switched bus fabric. Some of these are given in [4] [13] [40].

The global reconfiguration techniques can be classified into two categories:

- Row/column elimination
- Index mapping reconfiguration

4.2.1 Row/Column Elimination

In row/column elimination techniques, whenever a row/column contains at least one faulty cell, the whole row/column is bypassed, and a spare row/column is used as a substitute. Row/column elimination schemes are characterized by pre-defined interconnection resources, specified interconnection length, and simplicity of the interconnection networks. The constraint commonly imposed is that all rows (columns) of the reconfigured array have the same length. The advantage of this type of scheme is the simplicity of interconnection network, while high performance may not be guaranteed.

We illustrate this technique with an example. Consider a $(7+2)\times(9+3)$ fault tolerant processor array (FTPA). Some of the elements in the 7×9 array are faulty as shown in Figure 4.3(a). Shaded rows in the figure indicate spare rows and shaded columns indicate spare columns. If a random selection is made to replace rows and columns with spare rows and spare columns, then there is a possibility that at least some of the faulty elements are not covered. For instance, if we select row 1 and 3 and replace with spare rows and select columns 1,3 and 4 and replace with spare columns, then faulty cells (4,7) and (7,9) are not covered.

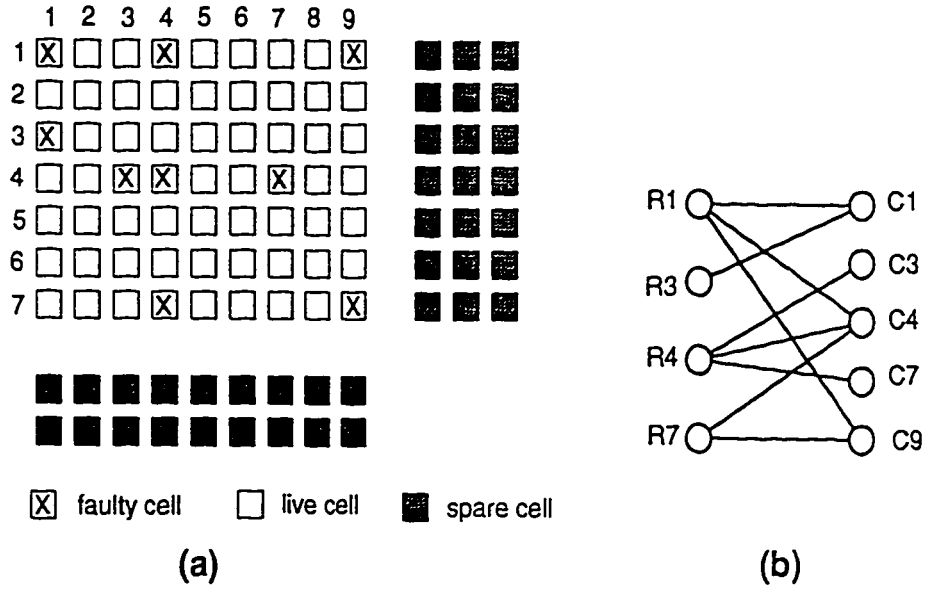


Figure 4.3: An example of row/column elimination technique (a) (7+2)×(9+3) FTPA with faulty cells (b) Bipartite description of fault distribution.

To solve the problem, the FTPA is modelled as a bipartite graph as shown in Figure 4.3(b). Now the problem reduces to finding a minimal cover. To find a minimal cover we have to select two vertices from the left and three vertices from the right such that elimination of these nodes and all edges connected to these nodes should eliminate all edges from the graph. Elimination of left vertices $R1$ and $R4$, right vertices $C1$, $C4$ and $C9$ and all edges connected to these vertices eliminates all edges from the graph. Selection of these vertices is proved to be an NP complete problem [24]. Many heuristics are proposed to obtain near minimal solution to row/column elimination problem [24] [51] [52] [53].

This scheme allows on-chip reconfiguration using simple controlling circuit as well

as simple algorithms, but spare utilization is very low [54].

4.2.2 Index Mapping Reconfiguration

In this approach, dimensions of the logical array and the physical array are usually pre-defined. A logical array is represented as $N_1 \times N_2$ matrix, whose elements correspond to the cells of the logical array. A physical array is denoted by $M_1 \times M_2$ matrix where each element indicates individual physical cell. Reconfiguration is achieved by mapping the logical cells onto fault-free cells of the physical array in such a way that each logical cell is assigned to a fault free cell only once. These approaches achieve good routing capability with pre-defined maximum interconnection length constraint and low overhead interconnection resources [4] [13]. In the following we describe some representative index mapping reconfiguration techniques.

The Full-Use-of-Suitable-Spare(FUSS)-Approach

The FUSS is based on the fault shifting strategy. Non-faulty PEs are shifted from regions with lesser faulty PEs to regions with more faulty PEs. To do this, it uses an indicator vector called the *surplus vector*, to guide replacement of faulty cells in a fault tolerant processor array.

In FUSS-C, the FTPA is an $M \times (N + C)$ array in which C is the number of spare

columns. First, the surplus is computed. Let f_i be the number of faulty cells in row i. The surplus vector is defined as

$$S = [s_1, s_2, ...s_M]^T$$

where

$$s_i = \sum_{j=1}^{i}(C - f_j) = C_i - \sum_{j=1}^{i} f_j$$

is the surplus on row i.

1. If $s_i > 0$ then the sum of spares in the row 1 through i is greater than the number of faulty cells in 1 through i, thus row i has extra cells available for use by faulty cells in row $i+1, i+2, ...M$.

2. If $s_i < 0$, then then row i has a deficit and needs to use available cells from row $i+1, i+2, ...M$.

3. If $s_M < 0$, then the number of spares is less than the number of faults. The array is not reconfigurable.

When the number of faulty processors in a row is greater than the number of extra spare columns, if through switching it is made possible to shift up or shift down some of the faulty cells, then the array can be reconfigured. This requires the shifting of the newly injected faults (virtual fault) in the adjacent row, where the fault is shifted. This is made possible with the help of a 3×3 switched bus interconnection as shown in Figure 2.4 .

The process is illustrated with an example. Matrix A represents a $4 \times (4+2)$ array. The table given below shows a physical arrangement of cells where 1 represents a faulty cell.

x	1	2	3	4	5	6	s
1	0	1	1	0	0	0	0
2	0	1	1	1	0	0	-1
3	0	0	0	0	0	0	1
4	0	1	0	1	0	1	0

The surplus vector is obtained as follows. $s_1 = 0$ as surplus is zero in row one. $s_2 = -1$ as faulty processors are three and $i = 2$ and $s_1 = 0$. s_3 =extra columns - faulty processors in row $3 + s_1$ = 2-0-1 = 1. Similarly, $s_4 = 0$. If s_4 is less than zero, then reconfiguration is aborted.

Then fault shifting is performed as shown in the table given below, such that $s_1, s_2, ..., s_n$ are all zero. d denotes a shift down and a u denotes a shift up. The reconfiguration is executed as follows

Scan array upwards : When $s_i < 0$ shift a number equal to $\mid s_i \mid$ of unavailable row i to available cells in row $i + 1$: so it is reset to 0 when all $\mid (s_i) \mid$ cells are shifted successfully. In the example $s_2 = -1$. Thus, one cell must be shifted from row 2 to row 3: cell (2,2), is shifted down to cell (3,2) which assumes the status code of d, s_3 is readjusted to 0.

Scan array downwards : When $s_i > 0$ shift a number equal to $\mid s_i \mid$ of unavailable

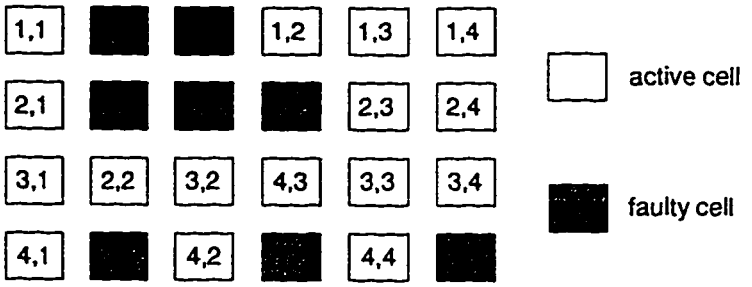


Figure 4.4: Final configuration obtained using FUSS.

rows $(i+1)$ to available cells in row i : so it is reset to 0 when all $\mid (s_i) \mid$ cells are shifted successfully. In the example $s_3 = 1$. Thus, one cell must be shifted from row 4 to row 3: cell (4,3), the only possible choice, is shifted upto cell (3,4) which assumes the status code of u, s_3 is readjusted to 0.

x	1	2	3	4	5	6	s
1	0	1	1	0	0	0	0
2	0	1	1	1	0	0	0
3	0	d	0	u	0	0	0
4	0	1	0	1	0	1	0

From the table given above the interconnection pattern can be found. It is as shown in Figure 4.4. The algorithm ensures 100% fault coverage, under the assumption that all types of interconnection are possible. With 3 x 3 interconnection resources 90-98% survivability is reported.

Some of the short comings of the algorithm are

- The algorithm assumes any kind of interconnection between two adjacent rows is possible. Therefore, the switch interconnection network should have any number of non-intersecting interconnection paths. For example, this technique is not suitable for 2×3 and 3×2 switch bus interconnection resources as it is prone to failures due to lack of interconnection resources.

- The maximum delay between the adjacent PEs can be established only after reconfiguration. No limits are set. Hence, clock period has to be more than the maximum delay between processors in the array.

- As the size of the array increases, not only does the reconfiguration complexity increases, but also the interconnection delay might become impractical.

- It cannot handle switch failures.

Rule-Based Reconfiguration Approach

The rule-based approach is based on reconfiguration using generic rules applicable to elements in the physical array for 3×3 interconnection resources. In this approach initially no logical cells are assigned to physical elements. The assignment of logical cell indices to physical element indices is based on the concept domain limits of a logical cell defined by variable domain approach (Section 2.2). However, not all the logical cell indices are assigned to physical element indices. To assign the remainder of the logical indices we use reconfiguration rules.

A physical cell may exist in the domain of more than one logical cells.

The approach consists of two phases : the global assignment phase and the rule-based assignment phase. Definitions necessary for explanation are given below. In the *Global Assignment Phase* the algorithm calculates the maximum dimension of a logical array and determines the 1st domain of each logical cell. The domains are variable and are formed from the faulty array. Let the size of the physical array be $L \times M$ and the size of the logical array be $N \times N$, where $N \leq L$ and $N \leq M$. Each row range has a top and bottom bounds. The top and bottom bounds of a row range are determined as follows. The top bound of row range 1 is set to 1, and the bottom bound of the row range N is set to L. i_B the bottom bound of row range i, is the minimum integer j which such that

$$\sum_{k=1}^{j} R_k \geq i \times N,$$

Where R_k denotes the number of fault free cells in physical row k. The bottom bound of the row range i is the top bound of the row range $i+1$. Since, the given 1×2 row interconnection resources have one horizontal track, only one physical row can be shared by two adjacent row ranges. If a physical row is shared by more than two row ranges, it is adjusted by shifting downward or upward bottom bounds of row ranges, so that only one physical row is shared by two adjacent row ranges. Columns ranges are defined similarly. The variable domains formed in a faulty 8×8 physical array to be mapped to 7×7 logical array is given in Figure 4.5. To obtain

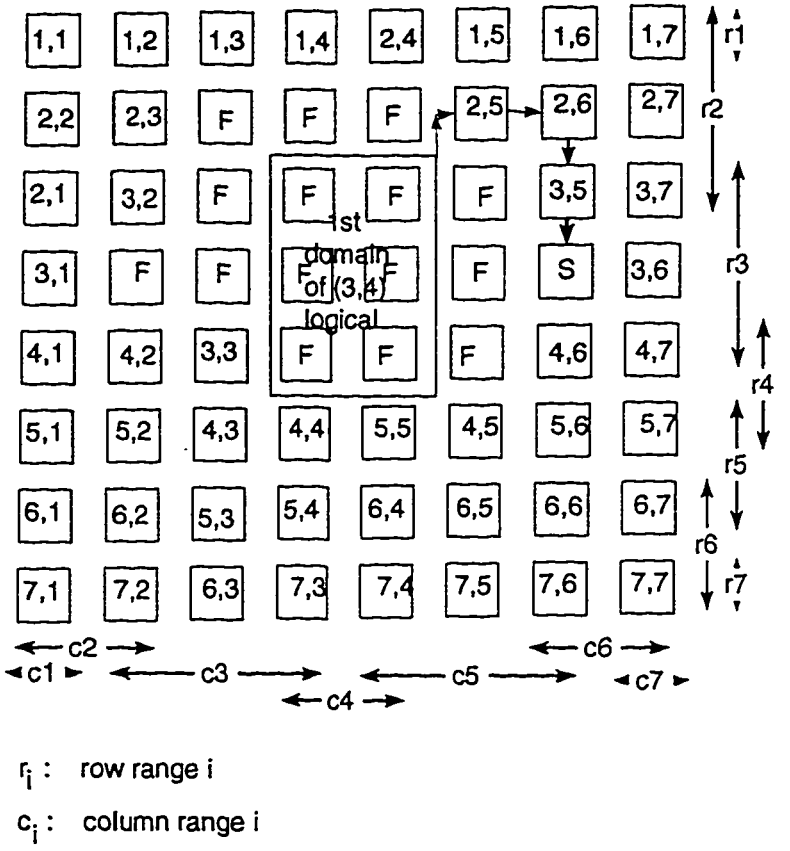


Figure 4.5: Reconfiguration of an 8 × 8 faulty array into a 7 × 7 fault free array.

row range $r1$ we evaluate the number of PEs in the first physical row. If it is less than 1×7 then, the row range is extended to physical row 2. There are 8 PEs in physical row 1, therefore, row range 1 contains physical row 1 only. Row range 2 contains the last physical row of the previous row range i.e. row 1 and other rows which satisfy the condition stated earlier. Here the sum of elements in row 1 and row 2 is less than 2×7. therefore, physical row 3 is also included in row range 2.

Once, row and column ranges are determined, each logical cell is assigned to a

fault free cell in the 1st domain. For the logical cell $(1,1)$, the 1st domain contains cells which are common to row range $r1$ and column range $c1$, i.e. physical cell $(1,1)$. The 2nd domain contains cells which are in the first domain of the logical neighboring cells. Maximum matching algorithm [55] is used to match logical cells to fault free cells in the logical domain. It may however, not be possible that all logical cells be assigned to fault free cells in the 1st domain. In Figure 4.5 logical cell $[3,4]_L$ could not be assigned to any of the physical cells in its 1st domain (cells in the rectangular box). Logical cells not assigned in the global assignment phase are assigned in the rule-based assignment phase.

Rule-Based Assignment

In the rule-based assignment phase, an assignment path is constructed based on the assignment rules. The construction of the assignment path for an unassigned logical cell is tried in its 1st domain. If no cells can be assigned to the unassigned logical cell in the 1st domain, cells in the 2nd domain are tried, and so on until kth domain. The 2nd domain contains the domains of the immediately neighboring logical cells. The parameter k is chosen considering the allowable size of the clustered faulty cells and prespecified intercell communication delay. To ensure a conflict free connection of the array, the assignment rules are checked during the construction of an assignment path. Assignment rules for 2×1 column resources and 1×2 row resources are given in Appendix A [21].

The neighboring cells of the logical cell $(3,4)$ are $(2,4)$, $(3,3)$, $(3,5)$ and $(4,4)$. Assignment of logical location $(3,4)$ in the 2nd domain and the subsequent assignment of the newly created unassigned logical location to other physical cells in their respective domains, leads to violation of reconfiguration rules, in all the four 2nd domains of logical cell $(3,4)$. Therefore, we select the 3rd domain of logical cell $(3,4)$. The 1st domain of logical cell $(2,5)$ contains three non-faulty cells $(2,6)_P$, $(2,7)_P$ and $(3,7)_P$. Subscript P represents physical cell. Logical location $(3,4)$ is transferred to physical cell $(2,6)_P$. Then logical location $(2,5)$ has to be assigned to another cell in its 1st domain. It is assigned to cell $(2,7)_P$. Similarly, logical location $(2,6)$ and $(3,5)$ are assigned to other non-faulty physical cells in their 1st domain. The chained move comes to an end when an unused non-faulty cell (called spare) is reached. The chained move is made in conformity with the reconfiguration rules.

Results obtained by this algorithm are claimed to be better than those obtained by the FUSS technique in terms of survivability. Moreover, the algorithm has the added flexibility of controlling the intermodule delays.

Some of the shortcomings of the algorithm are as follows:

- The variable domain approach is not suitable for 3×2 interconnection resources because the domain of a logical cell may not be safe for 3×2. Therefore, reconfiguration cannot proceed.

- It lacks the flexibility to handle failures due to switches.

The work proposed by [4] can be efficiently used for 3×3 interconnection resources. If 3×2 interconnection resources are used then validity of the initial array formed from the variable domains has to be checked. No work has been reported in [4] which is applicable to 3×2 interconnection resources.

4.3 Hierarchical Redundancy

In local redundancy techniques, if reconfiguration is not possible within a block or subarray, reconfiguration will fail. The tradeoffs in global redundancy are complexity of algorithm, interconnection delay and complexity. Hierarchical redundancy combines both techniques. First, it reconfigures locally within subarrays, if some of the subarrays cannot be reconfigured, they are eliminated in a global redundancy drive which reconfigures subarrays i.e. a faulty subarray is replaced by a spare subarray as shown in Figure 4.6. There could be different levels of hierarchical redundancy especially for very large arrays. Figure 4.6 shows two level redundancy.

A combinatorial analysis to obtain yield makes use of two probability functions. The occupancy function defines the probability of having exactly i boxes occupied, when r indistinguishable balls are randomly distributed into n distinguishable boxes.

Occupancy function is given by [15]

$$Q(r,n,i) = C(n,i)\sum_{v=0}^{i}(-1)^v C(i,v)[\frac{i-v}{n}]^r$$

where $C(n,i) = n!/[i!(n-i)!]$ is the total number of combinations of i elements out of n.

The other function is the probability of distributing k out of r indistinguishable balls into a box, given that the probability of a ball being placed into the box is t, is given by

$$P_d(r,k,t) = C(r,k)t^k(1-t)^{r-k}.$$

Yield of a block is given by Y_{bj}. It is obtained by using the following assumptions. Assume that the wafer has k defects and among them, c_1k defects are uniformly distributed and c_2k defects are clustered, where $c_1 + c_2 = 1$. Assume that r_1 defects out of c_1k distributed defects and r_2 out of c_2k clustered defects are distributed on the jth block. Among those $r_1 + r_2$ defects in the block, r defects are in cells and the rest of $r_1 + r_2 - r$ are in cells connecting channels. Then the yield of the jth block can be expressed as

$$Y_{bj} = \sum_{k=0}^{\infty}\sum_{r_1=0}^{c_1k}\sum_{r_2=0}^{c_2k}\sum_{r=0}^{r_1+r_2} P_w(k)P_d(c_1k,r_1,t_d)P_d(c_2k,r_2,t_c)P_d(r_1+r_2,r,t_e)Q_c(r)Q_w(r_1+r_2-r)$$

where $P_w(k)$ is the probability of array having k defects obtained from negative binomial distribution.

$P_d(c_1k,r_1,t_d)$ is the probability of exactly r_1 defects out of c_1k distributed defects

being located in the jth block, and t_d is the probability of one distributed defect being located in this block. $P_d(c_2k, r_2, t_c)$ is the probability of exactly r_2 defects out of c_2k clustered defects being located in the jth block, and t_c is the probability of a clustered defect being located in this block. $P_d(r_1 + r_2, r, t_e)$ is the probability of exactly r defects out of $r_1 + r_2$ distributed defects being located in the jth block, and t_e is the probability of a single defect being located in the cells of block j.

$Q_c(r)$ is the probability that by randomly distributing r defects into cells of a block, all defective cells can be replaced by spares. $Q_w(r_1+r_2-r)$ is the probability that by randomly distributing $r_1 + r_2 - r$ defects into cells containing channels, all defective channels can be replaced by spares.

Similarly, yield of block connecting channels is given by

$$Y_{wi} = \sum_{k=0}^{\infty} \sum_{r_1=0}^{c_1k} \sum_{r_2=0}^{c_2k} \sum_{x=0}^{t} P_w(k) P_d(c_1k, r_1, s_d) P_d(c_2k, r_2, s_c) Q(r_1 + r_2, m + t, x)$$

where $P_w(k)$ is the probability of array having k defects. $P_d(c_1k, r_1, s_d)$ is the probability of exactly r_1 defects out of c_1k distributed defects being located in the ith block connecting channel, and s_d is the probability of one distributed defect being located in this channel. $P_d(c_2k, r_2, s_c)$ is the probability of exactly r_2 defects out of c_2k clustered defects being located in the ith block connecting channel, and s_c is the probability of a clustered defect being located in this channel. $Q(r_1 + r_2, m + t, x)$ is the probability that $r_1 + r_2$ defects are distributed into x out of $m + t$ wires.

Yield of the array is given by

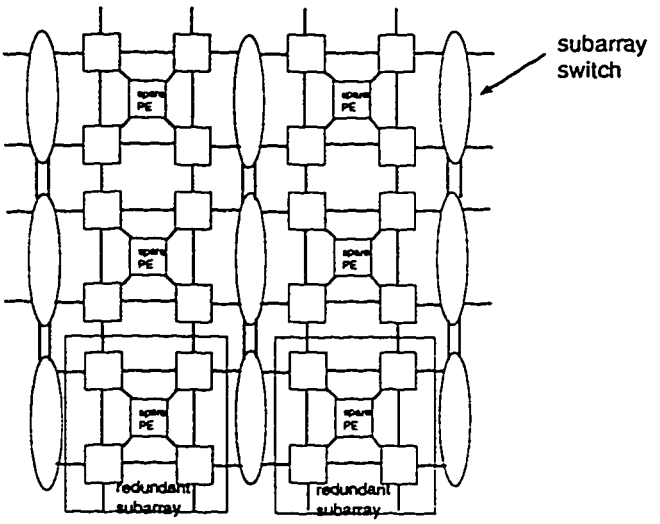


Figure 4.6: Two level redundancy in processor arrays.

Yield = $\prod_{all\ i}$ P(probability that the ith column of the blocks has at most one defective block) $\times$ $\prod_{all\ j}$ P(probability that the jth column of the blocks connecting channels has at most one defective channel).

Yield calculation for 12 $\times$ 12 mesh with block size 5 $\times$ 4 and 4 $\times$ 3 blocks are given in [15]. The final yield is equal to 0.172.

Large amount of redundancy can have negative effect on yield of the system [18]. One concern in this type of approach is, how many levels of redundancy should be adopted to obtain the best yield? Redundancy can be introduced within a processing element, within a subarray, and in between subarrays. A comparative analysis of various levels of redundancy is given in [56].

4.4 Hybrid Redundancy Schemes

Some approaches of yield enhancement of processor arrays combine the hierarchical and global reconfiguration techniques. To alleviate the problem of poor usage of spares due to rigid block boundaries in hierarchical techniques, architectures of global techniques are being used [10] [57]. Switched bus interconnection architectures with levels of redundancy as in hierarchical techniques provides flexibility to reconfiguration techniques to make better utilization of spares.

Column Based Hybrid Redundancy Approach

In this approach each column in a processor array is reconfigured separately. If all the columns are successfully reconfigured then the array is reconfigured successfully. Each column in the array is divided into overlapping blocks, with two spare elements, one on top and the other in the bottom of the block. If there are M non-spare elements in a column, and x non-spare elements in each block, then there are $\frac{M}{x} + 1$ spare elements. Block organization in columns is shown in Figure 4.7.

To reconfigure the array in the presence of faults, we first reconfigure within each block locally. If block local reconfiguration failed, then we continue reconfiguration in this block by borrowing the spares from the upper block first and then lower one block. Therefore, each original PE can be replaced by four spare PEs. An original

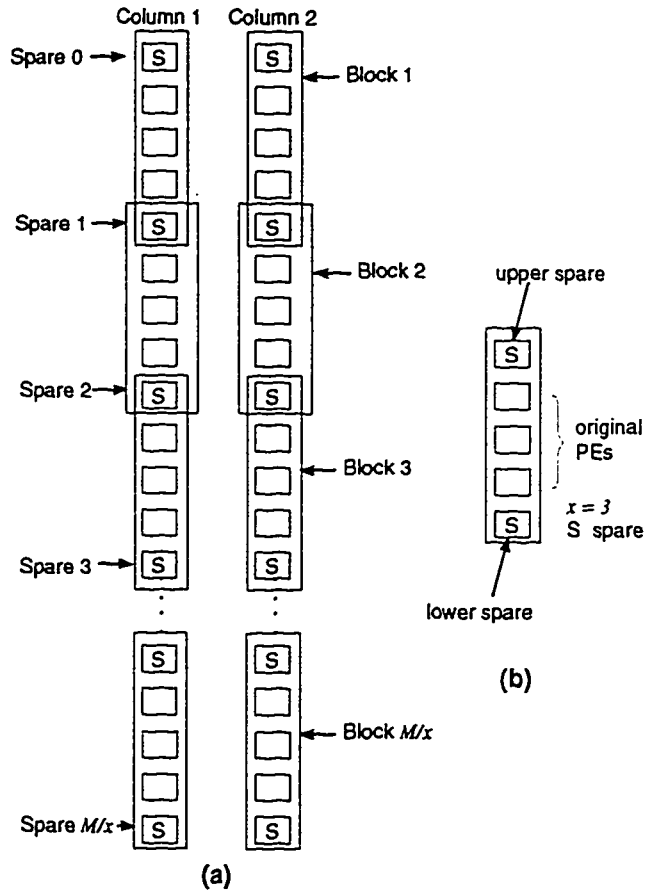


Figure 4.7: (The column based hybrid approach. a) Redundancy structure (b) Block organization

PE in a block is unavailable if it is faulty, and a spare PE is unavailable if it is faulty or it has been assigned to some other unavailable original PE. The reconfiguration procedure is shown in algorithm 4.1. SB(i,j) indicates the status of a block. SB(i,j) is 0 when the j^{th} block in i^{th} column is good. If SB(i,j)=1 then this block is faulty. Let UN(i,j) represents the number of unavailable PEs among $x+2$ PEs in a block. For reconfiguration the technique requires 3 vertical tracks for horizontal interconnection PEs and 1 horizontal track and 1 vertical track for vertical interconnection of PEs. Therefore, this architecture is a 3×2 interconnection architecture.

Yield of this architecture is more than yield of two-level redundancy. However, spares are not as efficiently utilized as in global reconfiguration techniques.

Algorithm 4.1 Hybrid Algorithm

Step 1:

block local reconfiguration
for each column i, $i = 1, \cdots, N$, do
for each block j, $j = 1, \cdots, M/x$, do
begin
case 1: condition: $UN(i,j) \leq 2$
action: SB(i,j)=0, replace the faulty original PEs by available spare PEs in the order of upper spare then lower spare.
case 2: condition: $2<UN(i,j)\leq 4$
action: SB(i,j)=1, replace the faulty original PEs by available spare PEs in the order of upper spare then lower spare. After replacement, modify UN(i,j) and record the locations of those unavailable original PEs which are not replaced by the spare PEs.
case 3: condition: $UN(i,j)>4$
action: system failure - exit
end

Step 2:

for each column $i, i = 1, \cdots, N$,do
for each block $j, j = 1, \cdots, M/x$,do
begin
if SB(i,j)=1 then
begin
if(spare available in upper one block) then
begin use it to replace the unavailable original PE left in current block, and modify UN(i,j).
begin
if(spare available in lower one block) and (UN(i,j)≠0) then
begin use it to replace the unavailable original PE left in current block, and modify UN(i,j).
end
if(UN(i,j)≠0) then system failure-exit.
else SB(i,j)=0 /*UN(i,j)≠0 → block failure → system failure*/
end

Partitioned Array Approach

In this approach a $M \times N$ two-dimensional array processor is partioned into $\frac{M}{L} \times \frac{N}{W}$ identical $L \times W$ subarrays, and each subarray (or block) is equipped with spare PEs as illustrated in Figure 4.8. The PEs are interconnected with 3×3 switch bus interconnections (shown in Figure 2.4(a)).

The reconfiguration is performed locally inside each subarray. Small target subarrays are created, and global reconfiguration is then accomplished by connecting the target subarrays. Since there is a single spare row or column between two adjacent subarrays, they can be used for both the subarrays. Due to the use of common spare resources, reconfiguration of two neighboring subarrays becomes coupled.

Reconfiguration of faulty elements is performed within each subarray separately.

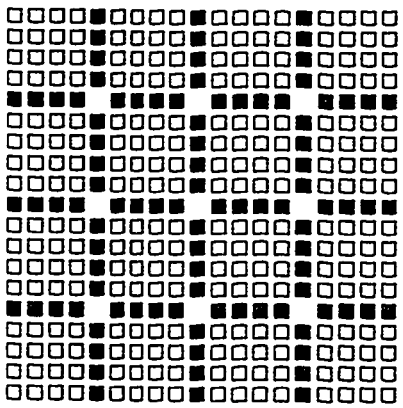


Figure 4.8: A partitioned array with spares indicated by black boxes.

Each subarray is scanned row wise and in each row, column wise i.e. in the order row 1 column 1 element, row 1 column 2 element, ..., row 1 column W element, row 2 column 1 ..., row L column W.

Depending on the number of faulty elements in a subarray, a faulty element is moved through a chain of replacements in one direction dictated by the following procedure. If $\mathrm{PE}(r,c)_B$ in a block is faulty then count the total number of faulty PEs at row r in subarray (i,j) and represent it with p. Similarly, the total number of faulty PEs in column c is represented as q. The replacement rules to substitute a faulty PE (r,c) by a good spare located either in row r or in column c are

(1) Case $p = q$, using the general rule: up $\rightarrow$ left $\rightarrow$ right $\rightarrow$ down sequence to borrow the spare PE to replace the faulty $\mathrm{PE}(r,c)$.

(2) Case $p > q$ using the column priority rule: up $\rightarrow$ down $\rightarrow$ left $\rightarrow$ right sequence to borrow the spare PE to replace the faulty PE (r,c).

(3) Case $p < q$, using the row priority rule: left $\rightarrow$ right $\rightarrow$ up $\rightarrow$ down sequence to borrow the spare PE to replace the faulty PE (r, c).

If for a faulty element a spare cannot be borrowed using the rules given above, then array reconfiguration fails.

4.5 Comparison of Existing Reconfiguration Techniques

A reconfiguration algorithm is optimal if the algorithm can find a solution when there exists one under prespecified constraints. At the end of production (i.e. static applications), yield improvement is the main concern. Successful reconfiguration rate and area overhead are some of the important factors. In static reconfiguration the algorithm complexity is of relatively low importance compared to the requirement of a successful reconfiguration rate as long as the time complexity does not grow exponentially with the size of an array [22].

Local reconfiguration techniques have low reconfiguration rate and have high hardware redundancy (100%, 50%). Such techniques are useful for yield enhancement of processor arrays which have critical interconnection delay requirement e.g. systolic arrays. Hierarchical reconfiguration techniques have moderate reconfiguration rate and moderate hardware redundancy. Global reconfiguration techniques have high

reconfiguration rate (90% to 99 %) and low hardware redundancy.

Survivability and algorithm complexity of some of the existing global reconfiguration techniques is given in Table 4.1 and Table 4.2 [9] [21]. ρ in Table 4.1 is *spare demand.* Spare demand is defined as the ratio of total number of faulty elements (F) to the total number of spare cells (S) i.e. $\rho = F/S$.

Reconfiguration Scheme	Algorithm Complexity	Survivability $\rho = .50$	$\rho = .75$	$\rho = .90$
Interstitial Redundancy	$O(n)$	0.43	0.35	0.27
Simple FUSS	$O(n^2)$	1.00	1.00	0.99
Rule-based Reconfiguration	$O(n^2)$	1.00	1.00	0.99

Table 4.1: A comparison of array survivability for different reconfiguration schemes.

Reconfiguration Scheme	Optimal switched bus	Scattered faults	Clustered faults	Spare redundancy No. of rows	No. of columns
Direct Reconfiguration	3×3	fair	bad	1	1
Simple FUSS	3×3	good	fair	1	1
Rule-based Reconfiguration	3×3	good	good	1	1

Table 4.2: Comparison of interconnection requirement, fault immunity and redundancy.

Simple FUSS and rule-based reconfiguration approach can achieve 99% survival rate for spare demand upto 0.9. This shows that spare utilization in these techniques is good. While simple FUSS cannot reconfigure an array in the presence of clustered faults with 3×3 interconnection resources, rule-based reconfiguration technique can

reconfigure the faulty array. However, rule-based approach assumes fault free status of switches and links in the array.

All the reconfiguration techniques given in the Table 4.2 require 3 × 3 switched bus interconnection resources for reconfiguration. Direct reconfiguration and simple FUSS techniques do not consider the reconfiguration resources required at the time of reconfiguration. Rule based reconfiguration technique was modelled on the basis of interconnection requirement. Direct reconfiguration and simple FUSS face a degradation in performance when interconnection resources are reduced. Rule-based approach [4] cannot reconfigure an array with 3 × 2 or 2 × 3 interconnection resources.

All the three approaches require the same amount of spare redundancy i.e. one spare row and one spare column for reconfiguration.

Performance of the rule-based approach is better than other techniques. It can handle clustered faults. Rules made for 3 × 3 interconnection resources can be modified easily to obtain rules for 3 × 2 and 2 × 3 reconfiguration resources.

4.6 Concluding Remarks

Local reconfiguration techniques and hierarchical reconfiguration techniques were aimed at developing reconfiguration strategies for very stringent interconnection length requirements. With advancements in VLSI/WSI technology global reconfig-

uration aimed at achieving maximum survivability were developed. However, these techniques do not necessarily impose restrictions on the length of interconnections between PEs and delays caused due to increased length, as it affects rate of successful reconfiguration. To improve utilization of spares in hierarchical redundancy while keeping the interconnection length within bounds, hybrid techniques were proposed. An important factor which did not receive much attention is faults in switches. Yield estimation model may give misleading results if faults due to soft switches is not considered [39] [10] [58]. However, yield loss due to faults in soft switches does not reflect yield of hard switches. Yield of restructurable laser fuse hard switches is reported to be as high as 99.999% [16]. However, large amount of interconnection resources, on one hand may achieve higher successful rate, but on the other hand require more hardware overhead and thereby increase susceptibility to faults [20]. Therefore, it is important to optimize interconnection resources used for reconfiguration in processor arrays. To optimize interconnection resources we require a platform which allows us to make estimation of yield for varying amounts of interconnection resources.

Chapter 5

The Proposed Framework

Local reconfiguration techniques requires very large amount of redundancy (at least 20%) and have limited capacity to tolerate faults. From the redundancy ratio indicated in [17] [18], yield loss may be too high.

Hierarchical reconfiguration schemes require moderately high spare redundancy as well as interconnection redundancy, especially for switching blocks.

Global reconfiguration techniques requires lesser percentage of spare redundancy as compared to other hardware reconfiguration techniques and have very high survivability. However, the minimum interconnection requirement to successfully reconfigure arrays under different possible fault distributions is 3×3 [21]. It is reported in [18] that when yields are dominated by interconnection area, yield improvement is not significant. Therefore, it is questionable whether 3×3 interconnection switch fabric is cost effective or not. This can be evaluated only when such circuits are

manufactured on commercial fabrication cites.

Due to the existence of limits on maximum and minimum redundancy required for manufacturing, there is an optimal spare redundancy and interconnection redundancy combination which maximizes yield. The optimal combination depends on size of the array and size of the processing elements. Since array size and processing element size are not homogeneous, the optimal combination of redundancy is application dependent. Therefore, a framework is needed which can simulate the survivability for a range of spare redundancy and interconnection redundancy combinations.

The framework for yield enhancement of processor arrays introduced in this thesis is modeled in three layers. The first layer is the core and all the rules for reconfiguration of switch bus are implemented in this layer. The second layer provides methods to search suitable spares in the processor array. In the third layer some reconfiguration algorithms are implemented. Apart from the three layers the framework also incorporates a module for the formation of initial solutions.

The rest of the chapter is organized as follows. In Section 5.1 we present an overview of the framework. Section 5.2 describes the scope of the core. Section 5.3 describes spare searching techniques (middle layer). Section 5.4 describes policy based algorithms for sequence of moves (top layer). In section 5.5 we discuss some methods to generate initial solutions. In section 5.7 we present some concluding

remarks.

5.1 Overview of the Framework

The framework for yield enhancement in processor arrays provides a platform to rearrange the logical cells on the processor array and a mechanism to search suitable spares for a given impaired element. Initially, spare elements are placed and an initial arrangement of logical cells on the physical array (initial solution) obtained. A good sequence of moves from an initial solution generates a final solution in which all the impaired elements are identified as bad elements.

The framework is divided into three layers. The lowest layer forms the core and provides the function of validating the assignment of logical elements to physical elements. The middle layer provides a mechanism to search spares for an impaired element in the array. The top most layer has different reconfiguration mechanisms at algorithmic level. The three layer model is shown in Figure 5.1.

The lowest layer deals with the rules for reconfiguration for different switch bus interconnection architectures. Rules proposed for 3×3 bus interconnected processor arrays [21] have been scaled down to obtain rules for 3×2 bus interconnected processor arrays (see appendix A). Rules for the vertical plane can be derived from the rules for the horizontal plane. Physical separation of the two planes necessitates

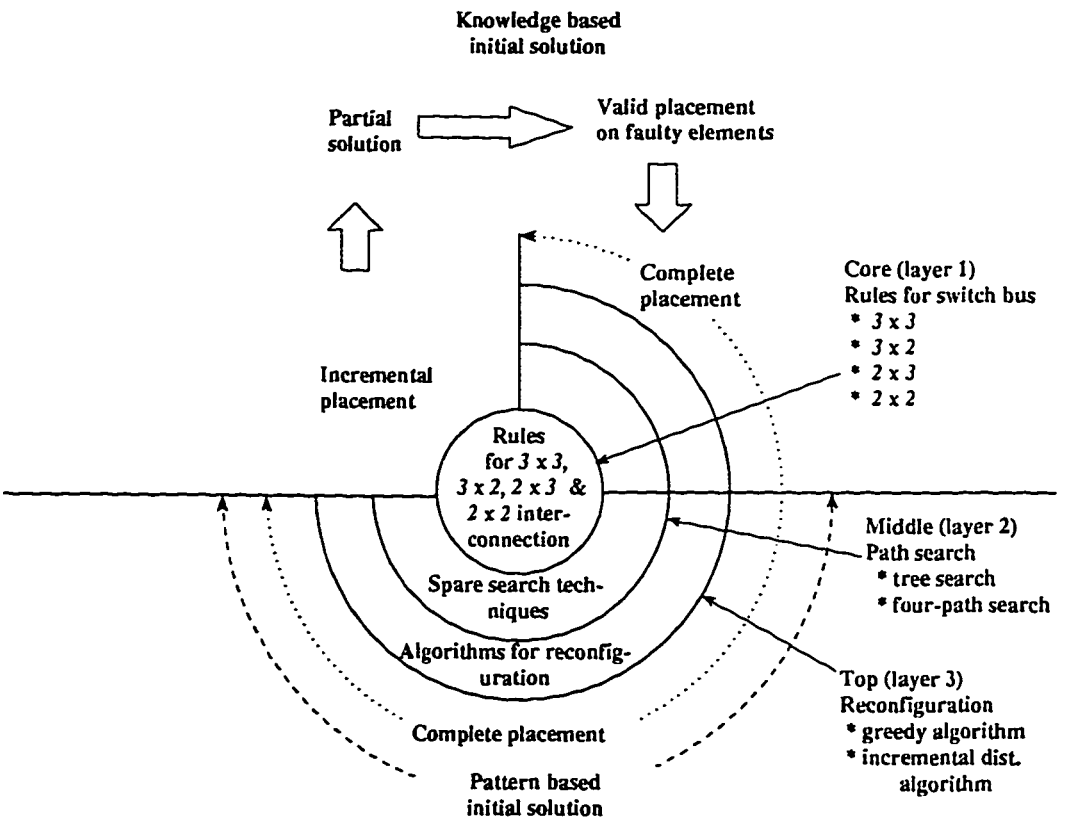


Figure 5.1: The three layer model of the framework.

independent implementation of rules for the horizontal and vertical planes.

The rules for different types of interconnection resources are different. From an initial solution with impaired elements to a final solution with all bad elements, the framework carries out the reconfiguration on any one type of bus interconnection resources (3x3 or 3x2). Rules for 3×3 are less restrictive than rules for 3×2 interconnection resources. Effectiveness of sequencing mechanisms may differ with interconnection resources in use.

At each reassignment, all the rules have to be checked [4]. If a a rule is violated, then the reassignment is invalid.

The middle layer deals with searching of spares to reassign impaired elements. After a spare is selected for an impaired element, then a move is made from spare to the impaired element and the rules are checked. If the move results in a violation of rules at any stage of reassignment, then we select another spare, if available, and try to make a new move.

In order to make an intelligent choice, the framework provides a mechanism to search spares at a given manhattan distance from a faulty element. The framework can also search all elements in the array and show all the valid spares and their associated distances from a faulty element.

The third layer deals with the sequence of moves made. Governed by a policy, the knowledge of valid moves that can be made at an instance of the reconfiguration process can be used to make the next move. The policy decides the sequence of

moves that are taken.

Two policies (schemes) for sequencing moves have been used in this framework. The first scheme is based on the greedy approach and the second scheme restricts reconfiguration to those elements which are at a specified manhattan distance from the impaired elements.

In the greedy approach we move from one impaired element to another only when the previous impaired element is successfully reassigned. The impaired element greedily seeks for a spare element irrespective of the reconfiguration resources it requires for reconfiguration. The first failure results in the failure of reconfiguration of the array. In the second approach we search for spares for an impaired element at a manhattan distance d. If there are none, then we try to move the next impaired element to a valid spare element at a manhattan distance d. When a pass in which all the impaired elements in the array are checked and reassigned, we make another pass with manhattan distance $d+1$ from the impaired elements that are left after the previous pass. We start with $d=1$ and go on uptill manhattan distance $M+N-1$ for an array of size $M \times N$.

Another factor on which the outcome of any reconfiguration mechanism may depend is the initial solution. The initial solution is a mapping of logical array on the physical array such that all the logical cells can be interconnected with pre-defined interconnection resources on the physical array. Physical elements to which no logical cells are assigned are called spare elements. The initial solution is defined as

the instance in the sample space of the FFA (fault free array) where we begin the reconfiguration process. The initial solution can be derived in two possible ways. These are pattern-based spare distribution and knowledge-based spare distribution. One in which knowledge of fault locations is not used and the other in which the knowledge of fault locations is used. When fault locations are not known then the initial solution is formed by distributing spares in a simple pattern as shown in figure 5.2. Such an initial solution is called *pattern based spare distribution.* In *knowledge based spare distribution* we form an initial solution by avoiding all the faulty elements. The unused non-faulty elements are called spares.

A placement in which all the logical cells are placed on the array and the rules are validated for all the logical elements is called a *complete placement.*

Starting the reconfiguration from a complete placement where the spare elements are in close proximity to faulty elements is more productive. But the location of faults in the array is not known a priori in pattern based spare distribution. With the assumption that the possibility of finding a PE faulty is same for all the PEs, we define a measure called *accessibility.* The measure accessibility and some procedures to obtain good initial solutions based on this measure, are given in Section 5.5.

An initial solution with good accessibility is shown in Figure 5.2. In the initial solution the spare row is the center row in the physical array. Similarly, the spare column is the center column in the physical array. The initial solution is valid for all types of bus interconnection resources. Other initial solutions are obtained from

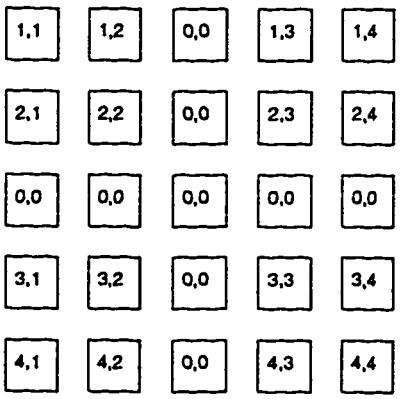


Figure 5.2: A simple initial solution. Square boxes show the physical locations and the logical locations are indicated in each box. (0,0) indicates a spare.

this initial solution. Any of the proposed reconfiguration algorithms in layer 3 can be used to obtain a final solution for a given fault pattern. The fault pattern determines the position of spares in the final solution. All physical elements that are not used in the final configuration are the new spare elements.

Knowledge-based formation of initial solutions can be obtained by using the variable domain approach proposed by Rhee [4]. In this approach bipartite matching of logical cells to non-faulty elements is obtained. The bipartite matching may leave some of the logical cells unassigned. Moreover, not all the matched elements may be valid if two switching tracks are used, rather than three, either in row or column or both. We call such a placement as an *incremental placement* because logical cells are placed one after the other. The incomplete solution obtained from bipartite matching is called *partial placement.* Unassigned logical cells can be assigned to faulty elements to obtain a valid complete placement.

The advantage of using the variable domain approach is that the initial solution obtained has very few impaired elements left. Moreover, the initial solution obtained have logical elements restricted to their safe domains (if 3×3 switch fabric is used) or near to safe domains for other interconnection fabrics.

5.2 Rules Platform - Layer 1

The basic foundation of the framework for yield enhancement of processor arrays is the rules platform. It provides a method to make a placement of logical cells on the array by avoiding congestion in channels during reconfiguration. This layer validates routability of different placements on the processor array. If the rules platform reports a conflict then there is congestion in atleast some channels on the processor array.

To cope with the objective of maximizing cost, the platform should provide means to assess the optimal interconnection requirement. Therefore, the rules platform provides the option of choosing the interconnection resources for each plane. The two possible choices are two tracks or three tracks per row/column of the processor array.

For three tracks per row (column), it is assumed that two tracks are vertical (horizontal) and one track horizontal (vertical) for horizontal (vertical) interconnection

plane.

For two tracks per row (column), it is assumed that one track is vertical (horizontal) and one track horizontal (vertical) for horizontal (vertical) interconnection plane.

If two tracks per row (column) is chosen for any plane, then duplicate assignment becomes very restrictive (see Appendix A). Therefore, if two tracks per row (column) are used in one plane then the other plane should have three tracks per column (row).

The rules platform assumes that the technology in use is RVLSI with laser programmable switches for reconfiguration. Therefore, the switch fabric is assumed to be fault free.

Difference in rules for the 3×3 and 3×2 interconnection resources, leads to different results for a sequence of moves. Many placements that are valid in 3×3 interconnection resources, are not valid for 3×2 interconnection resources due to less resources in the vertical plane. Therefore, efficacy of sequencing mechanisms may differ from one type of interconnection resources to another.

In order to make efficient utilization of interconnection resources, reconfiguration schemes should check validity of the current placement and make a good decision regarding the next placement. In order to aid reconfiguration techniques in making a good decisions regarding the next move, utilities are provided in layer -2. The following section describes spare searching utilities for reconfiguration in processor arrays.

5.3 Spare Searching Techniques - Layer 2

To obtain a valid sequence of reassignments from a spare element to an impaired element, we have to search all the path threads from the spare element to the faulty element. In the context of the framework the string of elements from a spare element to an impaired element, in which each element in the string is physically adjacent to the immediate neighbor in the string is called a *path thread.* If the length of a path thread is equal to the manhattan distance between the spare and impaired elements, then the path is a *direct path.* Path threads with length greater than the manhattan distance are called *detours.* A search in which all the moves are direct paths is called a *directed search.*

By directing the search from a spare PE to a faulty PE we restrict the search to a small portion of the processor array, indicating that the possibility of a successful reconfiguration of the faulty PE is reduced. But, as the number of reassignments of logical locations from source element to the destination element increases, the possibility that it results in a successful reconfiguration, decreases. Therefore, we expect that a directed search would perform as good as search using detours.

To make a move, all the logical cells on the elements in the path thread have to be reassigned to the neighboring elements along the path threads from spare element to the impaired element. First, a complete move is made then the rules are checked for every element with a logical cell mapped on it. If none of them causes a conflict,

the spare is assigned to the impaired element. If not, then placement on the array before the move is restored and another path thread from the spare to the impaired element is checked.

One question that comes to mind is how many paths should be checked. If there is a spare element and an impaired element in the array for which the difference in physical column index is m and the difference in physical row indices is n then there are $\binom{m+n}{n}$ paths with distance $m+n$.

If a conflict is encountered at an element on a path thread, then other path threads passing through that element are prone to failure. Therefore, the part of the tree at an element causing conflict can be pruned.

An exhaustive search of the path tree is very time consuming. Moreover, in most of the paths there will be more than two bends. Each bend is likely to cause a duplicate assignment (see section 4.2). Thereby, making reconfiguration more difficult. Therefore, we provide an alternate searching technique. In this technique, at most four path threads are checked. The four path thread are shown in figure 5.3(c).

If the physical row indices or physical column indices of the spare and impaired element is same then there is only one path thread. This is shown in figure 5.3(a). If the physical row indices and physical column indices of the spare and impaired element are not same then there are more than one path threads. If the difference between the physical row indices or physical column indices of the spare and impaired

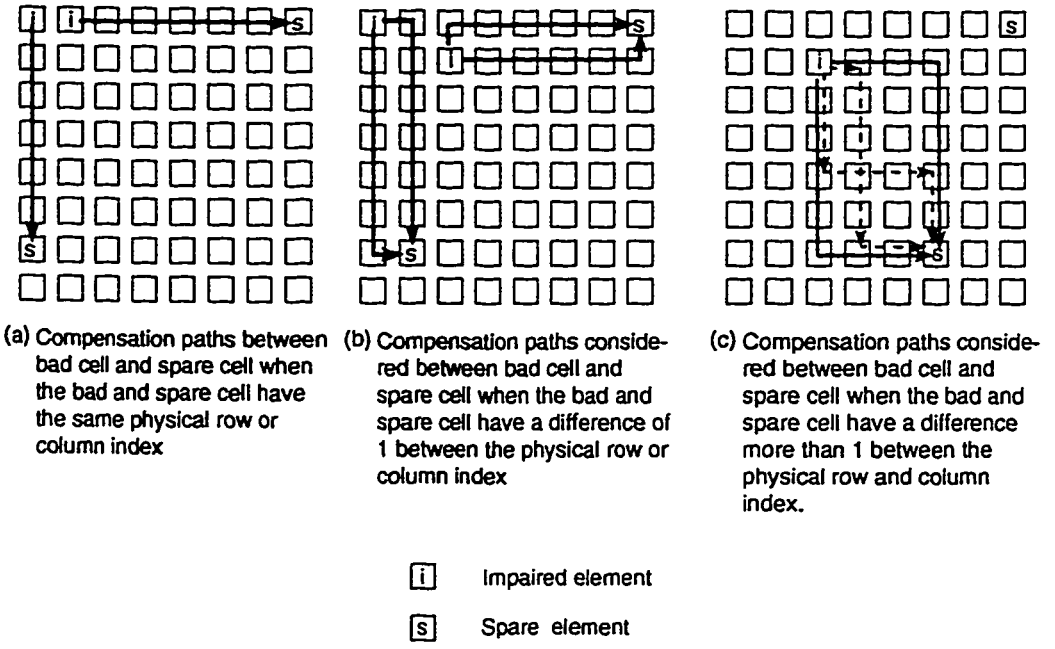


Figure 5.3: Compensation paths for different spare and impaired elements positions.

element is equal to one then, we consider the paths threads with one bend as shown in figure 5.3(b). Only two path threads exist between any two elements with one bend. If the difference between the physical row indices of the spare and impaired element is more than two and the difference between the physical column indices is also more than 2, then there are many paths threads. In this case only four path threads are checked. The four paths threads are shown in Figure 5.3(c).

As the distance increases the number of path threads with that distance increases. However, the probability of successful reconfiguration decreases for two reasons. The sequence of moves, will effect the final result. For larger trial distance, the chances of making a bad sequence of moves are expected to increase. The second reason

is that as we increase the trial distance, the probability of resource conflict is also expected to increase.

The cluster of impaired elements have to be checked after every move, as this region is most susceptible to a bad sequence of moves. One or two moves around it can block the reconfiguration of other impaired element.

Depending on the reconfiguration technique used and spare redundancy provided, we may require more exhaustive or less exhaustive searches. For this reason we have made utilities to make both the tree search as well as the four path search.

Using these utilities spare elements at a given manhattan distance from an impaired element can be easily searched. This is done by searching the spares at the given manhattan distance in a clockwise fashion. When a physical element at specified manhattan distance is spare then validity of the move from spare to the impaired element is checked. In Figure 5.4 for impaired element i and manhattan distance 2 then, element $(r4, c4)$ is checked first and then element $(r5, c7)$.

A complete search of the array for a spare element to which an impaired element can be moved is made by searching for spares at manhattan distance 2. Then distance is incremented and spare elements searched. The process is repeated till the complete array is searched. In Figure 5.4 for impaired element i sequence of spares searched are $(r4, c4)$, $(r5, c7)$, $(r3, c6)$, $(r7, c6)$, $(r8, c4)$, $(r6, c2)$ and $(r7, c8)$.

There are two modes of operation of the spare search. In the first mode the first valid move is made and the search discontinued. In the second mode all valid moves

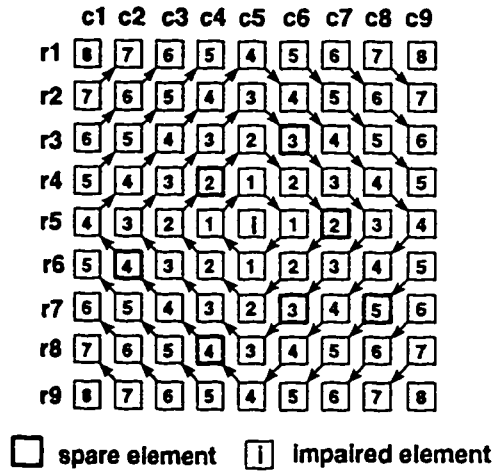


Figure 5.4: Clockwise searching of spare elements. Search begins from the elements on the left in the same row.

are searched and reported back with the original logical assignment passed back without a change.

Using the spare searching utilities reconfiguration techniques have been implemented in the third layer. In these techniques, reconfiguration begins from a complete initial solution which is valid (conflict free). Some elements are impaired. The sequence of moves taken to move impaired elements to spare elements, determines the final outcome. In the following section we present two reconfiguration techniques.

5.4 Policy Based Algorithms for Sequence of Moves - Layer 3

We begin the reconfiguration to obtain a valid solution for a fault pattern from an initial solution. From the initial solution we make a sequence of moves. The final outcome of the reconfiguration would have substituted all the impaired elements or may have some impaired elements left. In case all impaired elements are changed into bad elements, the reconfiguration is successful. Otherwise, the reconfiguration is unsuccessful.

Sequence of moves is defined as the sequence of *moves* of spare-impaired elements taken. At each step there is range of possible moves for all valid reassignments of impaired elements to spare elements. One move has to be selected. The selected move results in a new configuration.

When an impaired element is successfully moved to a spare element, a number of switches in the interconnection resources along the path of the move have to be re-configured. Due to reconfiguration of some of the switches, moves which were valid (invalid) earlier may become invalid (valid).

With every reassignment we progress towards a valid solution for the fault pattern. Every reassignment decrements the impaired elements count. As the number of spare elements decreases, the number of possible moves that can be taken at a step decreases. The decrease in range is attributed to a decrease in the number of spare

elements remaining and the effect of previous reassignments.

As the reconfiguration progresses, it becomes more difficult to reconfigure the remaining impaired elements. Therefore, the sequence of moves taken, becomes critical. Intuitively, it can be stated that a unique sequence of moves may not generate a valid configuration for every fault pattern. Therefore, we propose to make multiple attempts for reconfiguration, each with a different sequence of moves. The new initial solution should be a valid instance of the sample space of the fault free array and should be different from the previous initial solution. As we do not know the extent of the sample space of the fault free array and a method to transverse through all the samples, we propose to set the limits of initial solutions to a value depending on the resources for computation.

For each policy of sequence of moves, we attempt to obtain a valid configuration for a fault pattern from an initial solutions. When a valid configuration is not found we start again from another initial solution.

Two approaches are proposed for sequence of moves. The first is the greedy approach. In this approach an impaired element greedily seeks a spare element, irrespective of its distance from the spare element(cost). The impaired element is assigned to the closest spare element. We move to the next impaired element only after the previous impaired element is successfully assigned to a spare element.

The second approach is based on a policy called the incremental distance policy. In this approach we reassign an impaired element if it has a spare at a pre-specified

distance *d*. Therefore, the reconfiguration proceeds in passes. In each pass all the impaired elements are given a chance to reconfigure to a spare cell at a pre-specified distance.

The following sections describes the two approaches.

5.4.1 Greedy Algorithm

In the greedy approach we arrange the impaired elements in the form of a list. The spare elements are also arranged. The spare elements count should be greater than or equal to the impaired element count for the reconfiguration to proceed.

We begin the reconfiguration by finding all the valid spares that can be assigned to the first impaired element in the list (algorithm 5.1). We assign the impaired element to the closest spare element. In case there are no spare elements that can be assigned to the impaired element, the reconfiguration attempt for the given initial solution fails.

After the first impaired element is assigned successfully, the second impaired element is selected from the list. It is assigned to the closest spare element which results in a valid configuration. If there are no spare elements that are valid, the reconfiguration attempt fails.

The reconfiguration process is carried out for the rest of the impaired elements. If in the end, all the impaired elements are reassigned to spare elements successfully, the reconfiguration attempt succeeds.

Algorithm 5.1 Greedy Algorithm

Inputs : Faulty elements, Spare elements, Initial solution.

Output : Final solution.

Begin

Create *ImpairedElementsList, SpareElementsList.*

While (Not end of *ImpairedElementsList*) **Do**

Select next impaired element from *ImpairedElementsList*

If (Reassignment of impaired element to any spare element in the array succeeded)

update *ImpairedElementsList*;

update *SpareElementsList*;

Else

Reconfiguration Unsuccessful

Exit

EndIf

EndWhile

If (*ImpairedElementsList* is empty)

Reconfiguration Successful

EndIf

End

The reconfiguration process is illustrated with an example (Figure 5.5). We start the reconfiguration process from an initial solution valid for 3×3 switch bus as shown in Figure 5.2. Dashed boxes in Figure 5.5 are faulty elements in the array (e.g. $(2,3)_p$,$(3,3)_p$, etc). Some faulty elements are spare elements. Since, these elements are spare and faulty too they will not be used further in the reconfiguration process. Dashed boxes which have logical cells (written inside the boxes) are impaired elements (e.g. $(1,1)_p$,$(2,2)_p$, etc of initial solution). Dashed boxes to which no logical cells are assigned are bad elements (e.g. $(3,3)_p$).

In this example we have four impaired elements and four spare elements. The impaired elements are $(5,1)_p$, $(5,2)_p$, $(5,4)_p$ and $(5,5)_p$. The spare elements are indicated by **S** in Figure 5.5. They are $(1,3)_p$, $(3,1)_p$, $(3,2)_p$ and $(3,5)_p$.

The greedy algorithm in layer 3 orders the sequence of substitutions of spare elements to impaired elements. The impaired elements are searched in the array in a specified order. In the incremental distance algorithm we scan each row in the array left to right and rows from top to bottom (raster pattern). If we scan the array shown in Figure 5.5(b) in the specified pattern then the order of sequence of impaired elements are $(5,1)_p$, $(5,2)_p$, $(5,4)_p$ and $(5,5)_p$.

Once the array is scanned we start the reconfiguration process. Impaired elements are assigned to closest spares in the order specified earlier. First, a spare element is located for $(5,1)_p$. The closest spare element is $(3,1)_p$. The impaired element is assigned to spare element and the logical cell $(4,1)_L$ is moved as indicated in Figure

5.5(b). Next $(5,2)_p$ is assigned to $(3,2)_p$ as shown in Figure 5.5(b).

The third impaired element in the sequence is $(5,4)_p$. The closest spare to $(5,4)_p$ is $(3,5)_p$. Therefore, logical cell $(4,3)_L$ is moved from the faulty element as indicated in Figure 5.5(b). Finally, the last element $(5,5)_p$ is assigned to the available spare element $(1,3)_p$. The sequence of reassignments (move) is shown in Figure 5.5(c). The final configuration obtained has three duplicate assignments. The final configuration can be configured using 3×3 interconnection resources.

5.4.2 Incremental Distance Algorithm

Like the greedy approach, here we arrange the impaired elements in the form of a list. The spare elements are also arranged. The spare elements count should be greater than or equal to the impaired element count for the reconfiguration to proceed.

The reconfiguration of the array begins with the selection of trial distance *CurrentManhattanDistance = 1* (algorithm 5.2). All spare elements with trial distance *CurrentManhattanDistance* from the first impaired element are located. The impaired element is moved to a spare element which results in a valid configuration. After attempting to reconfigure the first impaired element to a spare element at *CurrentManhattanDistance*, we proceed to reconfigure the next impaired element to a spare element located at *CurrentManhattanDistance*. In the first pass we move

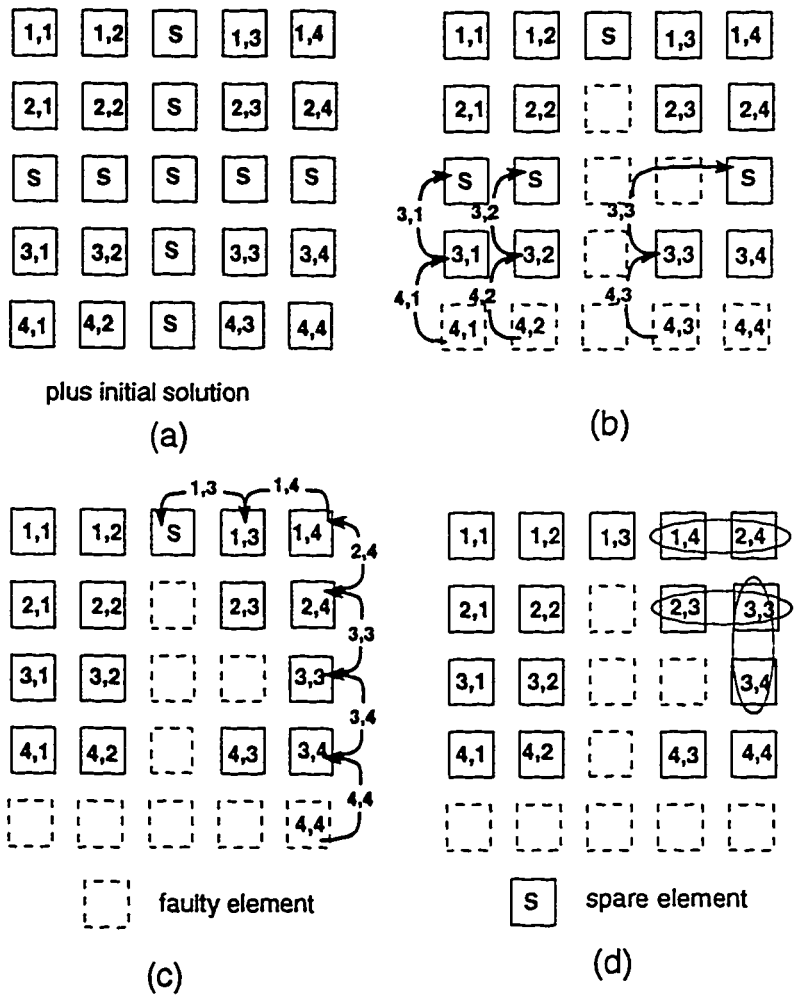


Figure 5.5: An example based on greedy algorithm.

all those impaired elements which are at *CurrentManhattanDistance* from the spare elements and which when moved results in valid configurations.

At the end of the first pass we check if there are any impaired elements in the array. If we find any impaired elements we make another pass with *CurrentManhattan-Distance = 2*. The first impaired element is selected and an attempt is made to move it to a spare element which results in a valid configuration. After attempting to reconfigure the first impaired element, we proceed to the next impaired element. Attempts are made to reconfigure the rest of the impaired elements to spare elements at *CurrentManhattanDistance*. If in the end of the pass, some impaired elements are left, we increment the *CurrentManhattanDistance* and make another pass. The process is repeated till all the impaired elements are moved to spare elements or the *CurrentManhattanDistance* exceeds *MaximumManhattanDistance*. The reconfiguration attempt is unsuccessful if *CurrentManhattanDistance* exceeds *MaximumManhattanDistance* and some impaired elements are still left in the array.

Algorithm 5.2 Incremental Distance Algorithm

Inputs : Faulty elements, Spare elements, Initial solution.

Output : Final solution.

Begin

Create *ImpairedElementsList, SpareElementsList.*

Set *CurrentManhattanDistance = 0;*

While (*CurrentManhattanDistance* < *MaximumManhattanDistance*)

Increment *CurrentManhattanDistance*;

While (Not end of *ImpairedElementsList*) **Do**

Select next impaired element from *ImpairedElementsList*

If (Reassignment of impaired element to any spare element

at a distance *CurrentManhattanDistance* succeded)

update *ImpairedElementsList*;

update *SpareElementsList*;

EndIf

EndWhile

If (*ImpairedElementsList* is empty)

Reconfiguration Successful

EndIf

EndWhile

Reconfiguration Unsuccessful

End

The reconfiguration process is illustrated with an example (Figure 5.6). We start the reconfiguration process from the initial solution shown in Figure 5.6(a). Dash boxes in Figure 5.6 are faulty elements in the array (e.g. $(2,3)_p$,$(3,3)_p$, etc). Dash boxes which have logical cells (written inside the boxes) are impaired elements (e.g. $(1,1)_p$,$(2,2)_p$, etc of initial solution). Dashed boxes (faulty elements) which do not have any logical cells are bad elements (e.g. $(3,3)_p$).

In this example we have four impaired elements and four spare elements. The impaired elements are $(5,1)_p$, $(5,2)_p$, $(5,4)_p$ and $(5,5)_p$. The spare elements are indicated by **S** in Figure 5.5 are $(1,3)_p$, $(3,1)_p$, $(3,2)_p$ and $(3,5)_p$.

The incremental distance algorithm in layer 3 orders the sequence of substitutions of spare elements to impaired elements. The impaired elements are searched in the array in a specified order. In the greedy algorithm we scan each row in the array left to right and rows from top to bottom (raster pattern). If we scan the array shown in Figure 5.5(b) in the specified pattern then the order of sequence of impaired elements are $(5,1)_p$, $(5,2)_p$, $(5,4)_p$ and $(5,5)_p$.

Once the array is scanned we start the reconfiguration process. Reconfiguration is performed in passes. In the first pass impaired elements are assigned to spare elements which are at manhattan distance 1 only. It may be possible to have no spare elements at manhattan distance 1 from an impaired element. In such a case the impaired element is overlooked and the next impaired element in the sequence tried. In this example none of the impaired elements have a spare element at manhattan distance 1 (See Figure 5.6(b)).

When all the four impaired elements are tried and some impaired elements exist in the array we begin a new pass and search spare elements at manhattan distance 2 from the impaired elements. In Figure 5.6(b) impaired element $(5,1)_p$ has a spare element $(3,1)_p$ at manhattan distance 2. Subsequent assignment of impaired element to spare element does not lead to any conflict. Therefore, $(4,1)_L$ is moved

as shown in Figure 5.6(b). The next impaired element in the sequence $(5,2)_p$ has a spare at manhattan distance 2. The reassignment does not lead to a resource conflict. Therefore, $(4,2)_L$ is moved as indicated in Figure 5.6(b). The third element is the sequence $(5,4)_p$ does not have any spare element at manhattan distance 2. Therefore, it is overlooked. We try the next impaired element $(5,5)_p$. $(5,5)_p$ is successfully assigned to spare element $(3,5)_p$. $(5,5)_p$ is the last impaired element on the array, if it the array scanned in raster pattern. This marks the end of this pass. Since, some impaired elements still exist we start a new pass with a new manhattan distance. A new pass is made and spare elements are searched for the left out impaired element $(5,4)_p$ at manhattan distance 3. Since, no spare exist more passes are made. In the pass with manhattan distance 6, spare element $(1,3)_p$ is located for impaired element $(5,4)_p$. Logical cell $(4,3)_L$ is moved as indicated in Figure 5.6(c). The final configuration shown in Figure 5.6(d) has only one duplicate assignment.

This section presented techniques to reconfigure processor arrays in the presence of faults in the array. In both the techniques we start the reconfiguration from an initial solution. In the following section techniques for obtaining initial solutions are presented. Initial solutions can be obtained in two ways: without using the knowledge of fault locations i.e. pattern based spare distribution and using knowledge of fault locations i.e. knowledge based spare distribution.

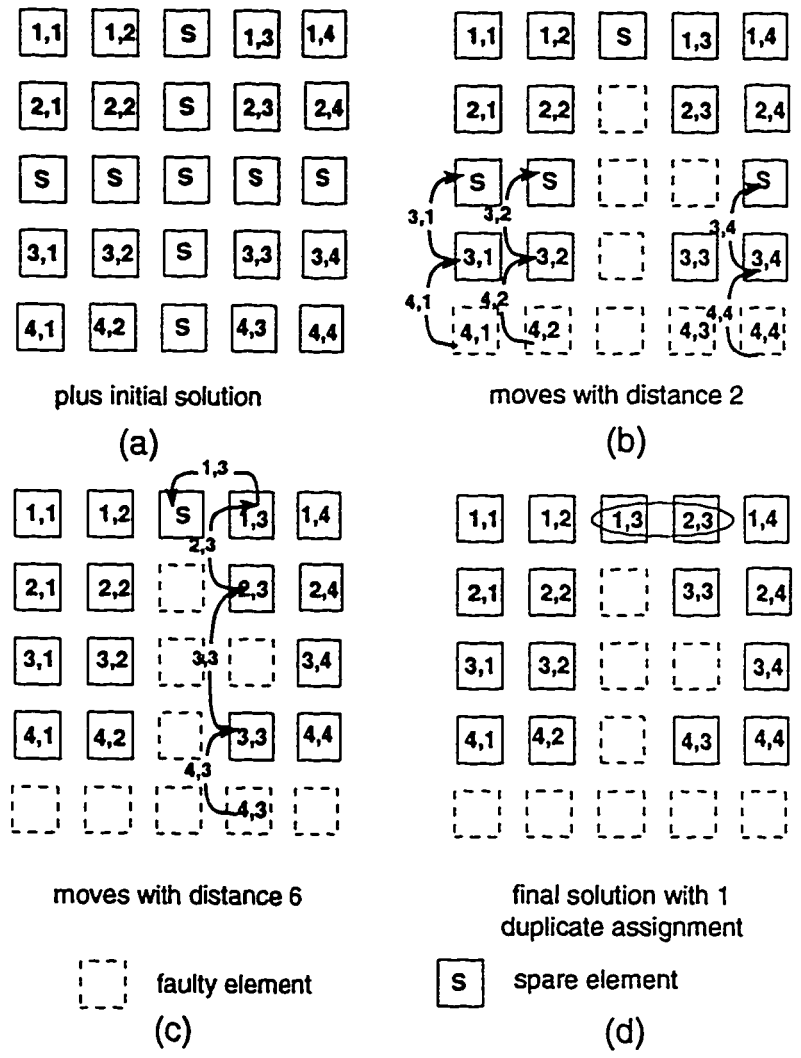


Figure 5.6: An example based on incremental distance algorithm. All steps taken are with indicated manhattan distance.

5.5 Formation of an Initial Solution

As the size of the sample space of the FFA increases it becomes more difficult to successfully reconfigure different faulty arrays. A comprehensive search of every possible sample of the FFA is time consuming and difficult to formulate. Therefore, we propose to reconfigure from a few selected initial solutions.

An initial solution represents an initial arrangement of logical cells on the physical array. It is also an instance in the sample space of the FFA. Therefore, the reconfiguration problem is to find another instance in the sample space of the FFA in which all the faulty elements are substituted, starting from the initial solution.

The initial solution plays an important role in the success or failure of the reconfiguration attempt. Therefore, there are two questions to be answered. Which initial solutions do we use and how do we form these initial solutions ?

To answer the first question we take a closer look at the global reconfiguration schemes. There are two approaches. In one approach the location of spare elements is fixed e.g. a row/column, a row and a column, two rows/columns etc [13]. In the other approach nothing is known about the location of spare elements. The complete array is considered as a set of PEs onto which the logical cells have to be mapped. The non-faulty elements that are left out after logical cells are mapped to the array are considered as spare elements [4].

In the first approach there is no prior knowledge of the location of faulty elements.

The location of spare elements are fixed and all the logical cells are configured on the non-faulty elements in a simple form. Therefore, the reconfiguration problem is to restructure the current assignment of logical elements such that impaired elements are moved to spare elements. Such type of initial solutions are termed as pattern based, and spare distribution as pattern based spare distribution. The reconfiguration problem is called initial solution based reconfiguration (see figure 5.1).

Success of the reconfiguration attempt, in this approach, depends on the relative locations of impaired elements and spare elements. The ease of reconfiguration of an impaired element depends on how far the spare element is placed and the arrangement of logical cells in between the impaired element and the spare element. Lesser the distance between impaired element and spare element, the lesser are the number of reassignments, therefore a greater chance of success. But, the arrangement of logical cells can only effect reassignment of logical cells along the path thread. Therefore, we define a measure of goodness of spare distribution based on a metric called *accessibility*. The pattern based distribution of spare elements and the subsequent formation of the initial solution is given in section 5.5.1.

In the second approach, the initial solution is formed by using the knowledge of location of faulty elements. This is done by using bipartite matching [21]. The configuration is done incrementally, one logical cell at a time. We call such placement as incremental placement.

The initial solution has almost all the faulty elements eliminated. However, not

all the logical cells may be placed. Therefore, the initial solution is incomplete. To complete the initial solution the logical cells are placed on any valid physical location (generally faulty). Since some logical cells are not assigned, there are some unused non-faulty elements in the array. The unused non-faulty elements are called spare elements. The spare elements can be located anywhere in the array. After assigning the unassigned logical cells to faulty elements, the reconfiguration problem reduces to initial solution based reconfiguration.

By making a bipartite matching or incremental placement of logical cells to non-faulty physical elements while keeping the map of the range of a logical cell in the physical elements domain restricted to the safe domain of 3×3 interconnection resources, we preserve the structure in the partial placement. A structured partial placement has room for reconfiguration as there are few duplicate assignments made. The impaired elements that are left after assignment of logical cells on faulty elements (formation of complete solution) are very few. The impaired elements have to be reconfigured to obtain the final placement.

To obtain different initial solutions we can inject dummy faults in the array. The number of faults injected would be the difference in the number of redundant elements and faulty elements in the array. By injecting dummy faults randomly in different locations we obtain different initial solutions.

Details of formation of initial solution using knowledge based spare distribution is given in section 5.5.2.

5.5.1 Pattern Based Spare Distribution

Starting a directed search from a point in the sample space where the spare elements are in close proximity to a faulty element is more productive. But the location of faults in the array is not known a priori. Recent studies have shown variations in defect densities depending on the location of the PE on the processor array (see section 3.4). However, exact measures of relative dependence are not reported so far. Therefore, we assume that the possibility of finding a PE faulty is same for all the PEs. Such assumption implies that a good spare distribution will have a spare close to every element in the processor array. Based on this criteria we define a measure called *accessibility.*

Definition : Accessibility is defined as the reciprocal of the average of manhattan distances between all physical elements in the array to their closest spare elements.

$$1/Accessibility = \frac{\sum_{i=1}^{M}\sum_{j=1}^{N}(min(\mid xp_i - xs_u \mid + \mid yp_j - ys_v \mid))}{total\ number\ of\ logical\ cells}$$

where, u ranges from $1, ..., M$ and v ranges from $1, ..., N$, (xp_i, yp_j) represent the physical elements in the array and (xs_u, ys_v) represents the spare elements in the array. □

To evaluate accessibility, we average the manhattan distance between each physical element and a spare element closest to it. The reciprocal of this number is

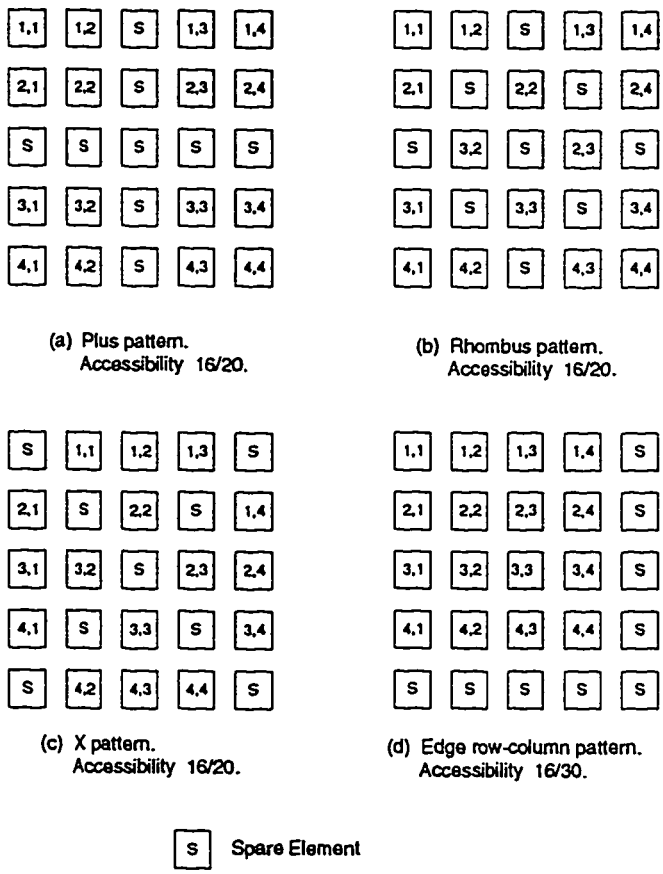


Figure 5.7: Initial solutions for 5 × 5 physical array and corresponding values of accessibility.

accessibility. Accessibility of spare distributions of arrays given in figure 5.7(a) is $\frac{16}{20}$, figure 5.7(b) is $\frac{16}{20}$, and figure 5.7(c) is $\frac{16}{20}$. Accessibility from physical elements to spare elements in an array with fifth row and fifth column of spare elements in figure 5.7(d) is $\frac{16}{30}$. Higher values of accessibility indicate a good spare distribution.

Having a measure to evaluate the feasibility of initial solutions, we can construct new initial solutions. To form a new initial solution, first we obtain distribution of spares which have a relatively higher value of accessibility. Such distribution can be based on intuition or can be a random selection. After all spare locations on the physical array are finalized, we have to map the logical cells on the physical array such that none of the spare locations are used. Mapping the logical cells on the physical array has to take the validity of the new solution into consideration (rule checks). Mapping logical cells directly would make the process of obtaining an initial solution non-deterministic. Therefore, we adopted an indirect approach. First, we begin from any of the previously obtained initial solutions. We then mock the spare locations of the new spare distribution as faulty elements. Using previously described reconfiguration strategies we obtain a final solution to the faked fault distribution. Final solution of a mocked fault pattern may not have eliminated all the mocked impaired elements. However, if accessibility of the spare distribution is comparable to accessibility of other initial solutions, then we can retain the final solution. The final solution can be considered as an initial solution as it is another instance of the sample space of the FFA. However, if the number of mocked impaired

elements left unmoved is large, then the final solution (instance in sample space of FFA) would be close to the initial solution (another instance in sample space of FFA) in the sample space of the FFA. Thus, we would be making repetitive search in some regions of the sample space.

Starting from an initial solution we converge towards a final solution eliminating the mocked faulty elements. The distribution of mocked faulty elements, which is the spare distribution, can be made randomly or using some defined logic. In randomly generated spare distributions, spares are randomly selected. Those distributions that have accessibility more than a prespecified value are processed further.

In logic based spare distribution, we form patterns out of the spare distributions. Generally, there is an underlying logic for most of the patterns. For example, the spare distribution shown in Figure 5.7(a) called plus pattern has spare elements in the center row and spare elements in center column. A procedure to automate spare distribution can be easily formulated. Spare distribution of rhombus pattern and X pattern is more complex. Procedures in the form of algorithms to generate these distributions for all sizes of arrays are given in Algorithm 5.3 and Algorithm 5.4.

Regeneration of logic based spare distributions is definitive while regeneration of random spare distributions is dependent on the random number generators. Therefore, logic based spare distribution is preferable.

Algorithm 5.3 Rhombus Spare Distribution

Inputs : *Rows, Columns, TotalNumberofSpares.*
Output : Spare Distribution.
Variables: $CurrentRow, CurrentColumn, CurrentSpareCount.$

Begin

Initialize $CurrentRow = 1$

$CurrentColumn = Columns/2$

While $(CurrentColumn <= Columns)$ **Do**

Label $(CurrentRow, CurrentColumn)_p$ spare.

Increment $CurrentRow, CurrentColumn, CurrentSpareCount$

EndWhile

While $(CurrentRow <= Rows)$ **Do**

Label $(CurrentRow, CurrentColumn)_p$ spare.

Increment $CurrentRow, CurrentSpareCount$

Decrement $CurrentColumn$

EndWhile

While $(CurrentColumn >= 1)$ **Do**

Label $(CurrentRow, CurrentColumn)_p$ spare.

Decrement $CurrentSpareCount$

Decrement $CurrentRow, CurrentColumn$

EndWhile

While $(CurrentRow >= 1)$ **Do**

Label $(CurrentRow, CurrentColumn)_p$ spare.

Increment $CurrentColumn, CurrentSpareCount$

Decrement $CurrentRow$

EndWhile

While$(CurrentSpareCount < TotalNumberofSpares)$**Do**

Generate randomly $CurrentRow, CurrentColumn.$

Label $(CurrentRow, CurrentColumn)_p$ spare.

Increment $CurrentSpareCount$

EndWhile

End

Algorithm 5.4 X Spare Distribution

Inputs : *Rows*, *Columns*, *TotalNumberofSpares.*
Output : Spare Distribution.
Variables: $CurrentRow, CurrentColumn, CurrentSpareCount.$

Begin

Initialize $CurrentRow = 1$

$CurrentColumn = 1$

While $((CurrentColumn <= Columns)$ &

$(CurrentRow <= Rows))$ **Do**

Label $(CurrentRow, CurrentColumn)_p$ spare.

Increment $CurrentRow, CurrentColumn, CurrentSpareCount$

EndWhile

Initialize $CurrentRow = 1$

$CurrentColumn = Columns$

While ($CurrentColumn >= 1$ & $CurrentRow <= Rows$) **Do**

Label $(CurrentRow, CurrentColumn)_p$ spare.

Increment $CurrentRow, CurrentSpareCount$

Decrement $CurrentColumn$

EndWhile

While($CurrentSpareCount < TotalNumberofSpares$)**Do**

Generate randomly $CurrentRow, CurrentColumn$.

Label $(CurrentRow, CurrentColumn)_p$ spare.

Increment $CurrentSpareCount$

EndWhile

End

In the following example we show the formation of an initial solution if the spare elements are arranged in the form of an X logic pattern. Initially, we obtain the spare pattern using Algorithm 5.4. The spare distribution is indicated with shaded boxes in Figure 5.8.

The initial solution for the plus pattern can be obtained easily by forming rows of logical cells on non-spare rows on physical array. The initial solution with plus spare

pattern in shown in Figure 5.8 step 0. The shaded boxes which are in the form of a X pattern are mocked faulty elements. Using greedy algorithm we reconfigure the array as shown in Figure 5.8 step 1 through Figure 5.9 step 8. The final configuration is an initial solution with X pattern of spare elements. This is shown in Figure 5.9 step 8.

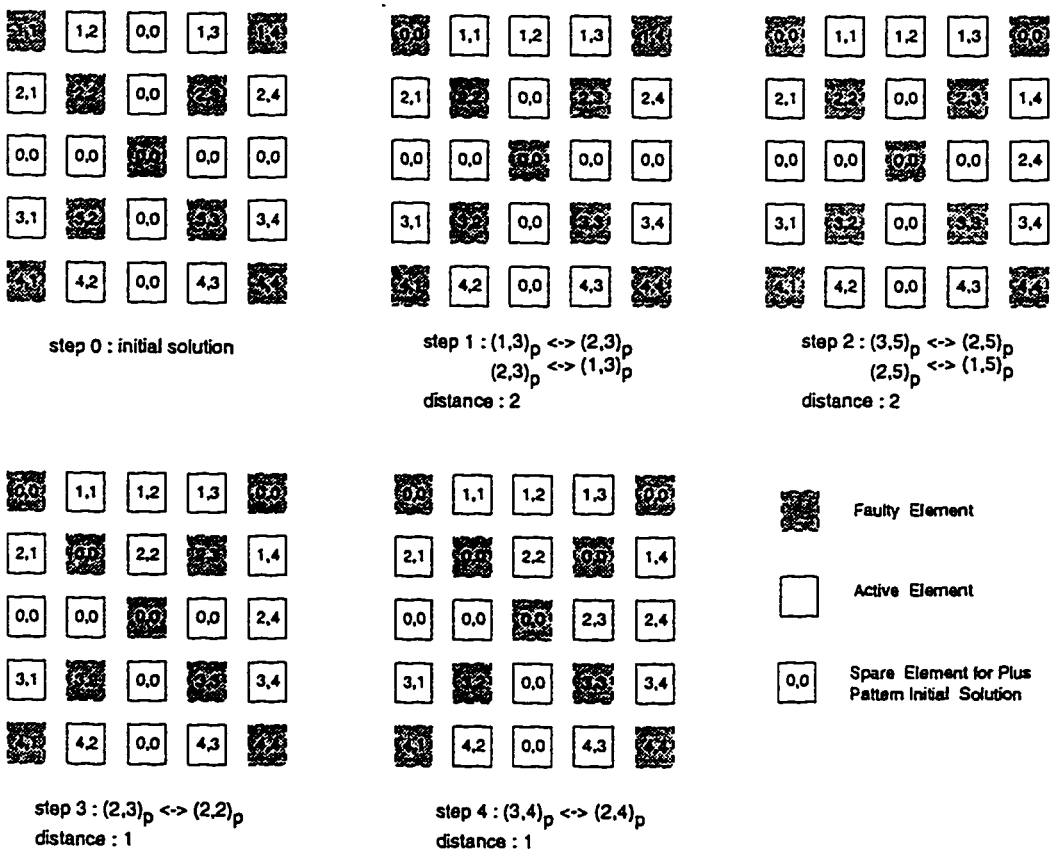


Figure 5.8: Formation of X pattern based initial solution using greedy algorithm.

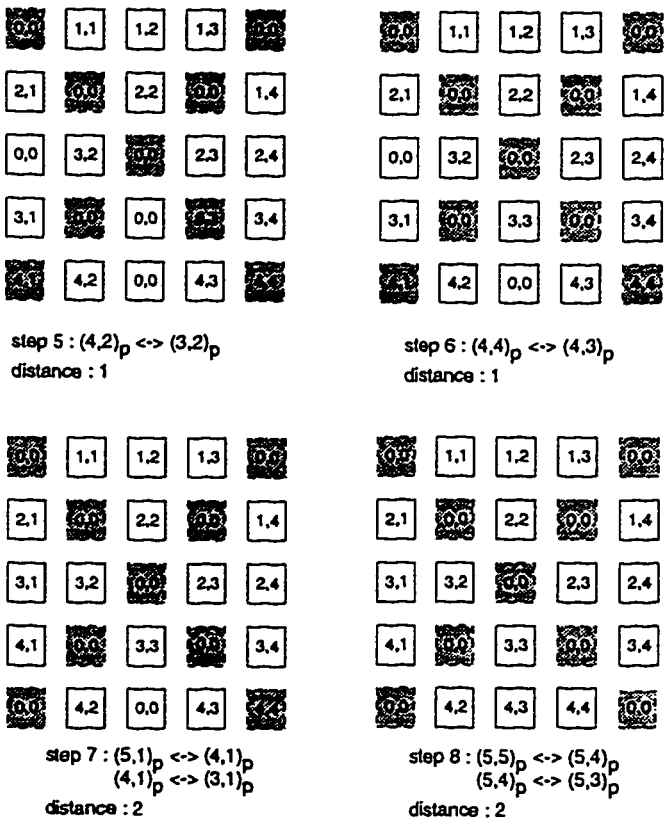


Figure 5.9: Continuation of the example based on greedy algorithm (Figure 5.8).

5.5.2 Knowledge Based Spare Distribution

In this approach the initial solution is formed by using the knowledge of location of faulty elements. This is done by forming variable domains as given in section 4.2.2. The variable domain of each logical cell gives the range of elements onto which the cell can be mapped and is safe for 3×3 interconnection.

Once domains for all cells is made a bipartite graph is constructed with logical cells as one set of vertices and the elements as the other. The map obtained from variable domains define the edges between the two sets of vertices. We can obtain a partial placement by using bipartite matching (not necessarily maximal) as in [21].

The configuration is done incrementally i.e. by placing logical cell by logical cell. Therefore, we call such placement an incremental placement.

The initial solution has almost all the faulty elements eliminated. However, not all the logical cells may be placed. Therefore, the initial solution is incomplete. To complete the initial solution the logical cells are placed on any valid physical location (generally faulty). Since some logical cells are not assigned, there are some unused non-faulty elements in the array. The unused non-faulty elements are called spare elements. The spare elements can be located anywhere in the array.

After assigning the unassigned logical cells to faulty elements, the reconfiguration problem reduces to initial solution based reconfiguration (see figure 5.1).

By making a bipartite matching or incremental placement of logical cells to non-

faulty physical elements while keeping the map of the range of a logical cell in the physical elements domain restricted to the safe domain of 3×3 interconnection resources, we preserve the structure in the partial placement. A structured partial placement has room for reconfiguration as there are few duplicate assignments made. The few impaired elements that are left after assignment of logical cells on faulty elements (formation of complete solution) are easily reconfigured.

To obtain different initial solutions we can inject dummy faults in the array. The number of faults injected would be the difference in the number of redundant elements and faulty elements in the array. By injecting dummy faults randomly in different locations we obtain different initial solutions. As the number of faults in the array increases the number of dummy faults that can be injected decreases. When the number of faults is equal to the number of redundant elements then only one initial solution can be formed.

The process of formation of initial solution and subsequent reconfiguration is explained with the aid of an example. To form initial solution using bipartite matching or incremental placement on a 7×7 physical array and logical array of size 6×6, with the given fault distribution in Figure 5.10(a), we form variable domains as given in [21]. We start counting the number of non-faulty elements in rows from the first row. Row range of logical row i is from the physical row in which $(i-1) \times 7$ non-faulty element count is met, to the physical row in which $i \times 7$ non-faulty element count

is met. For logical row 1 the range is from physical row 1 to row 1 ($r1$). For logical row 3 the range is row 2 to 4 as indicated by $r3$. Similarly, the column ranges are defined.

The elements that lie in the overlapped range of the logical row i and logical column j constitute the domain of the logical cell (i,j). Domain of logical cell $(2,5)_L$ are elements $(1,4)_P$, $(2,4)_P$, $(1,5)_P$, $(2,5)_P$, $(1,6)_P$ and $(2,6)_P$. Faulty elements $(1,5)_P$ and $(2,5)_P$ are not used.

Restricting the mapping of logical cells to physical elements in their respective domains, we make a bipartite matching. Partial placement for 3×2 switch bus interconnection obtained from bipartite matching is shown in Figure 5.10(b). The logical cell $(2,5)_L$ is still not placed. To obtain the complete placement the unassigned logical cell is placed on the faulty element $(1,5)_P$. The complete placement is shown in Figure 5.10(c). Using the rules for 3×2 and greedy reconfiguration algorithm we obtain the final configuration shown in Figure 5.11.

5.6 Complexity Analysis

Reconfiguration of a faulty processor arrays is done by using the three layers of the framework starting from an initial solution. The overall complexity of reconfiguration will be determined by the complexity of formation of initial solution and that of reconfiguration process to obtain the final configuration.

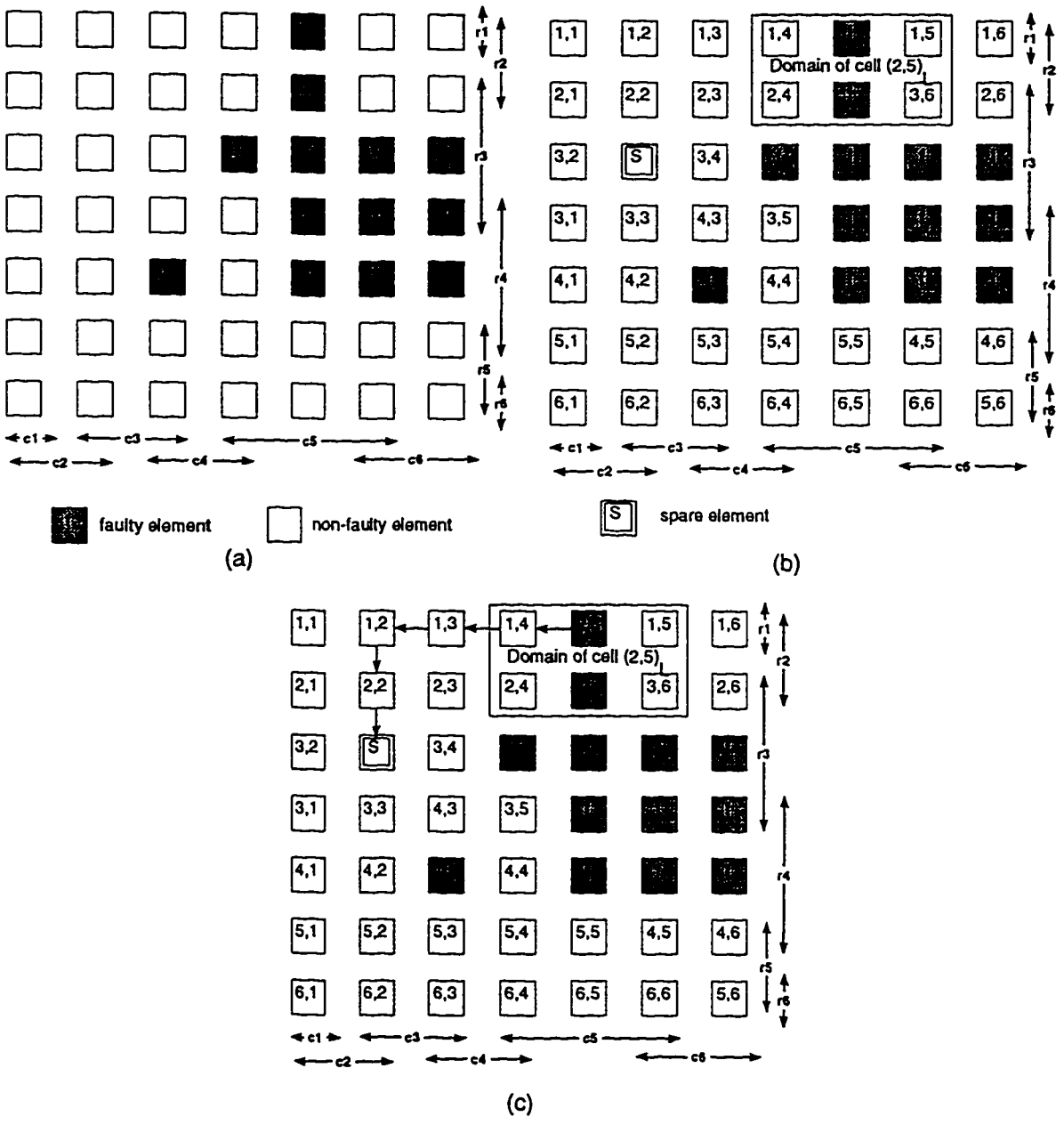


Figure 5.10: Formation of knowledge-based spare distribution. (a) Formation of variable domains (b) Partial placement obtained from bipartite matching (c) Complete placement obtained by placing cell $(2,5)_L$.

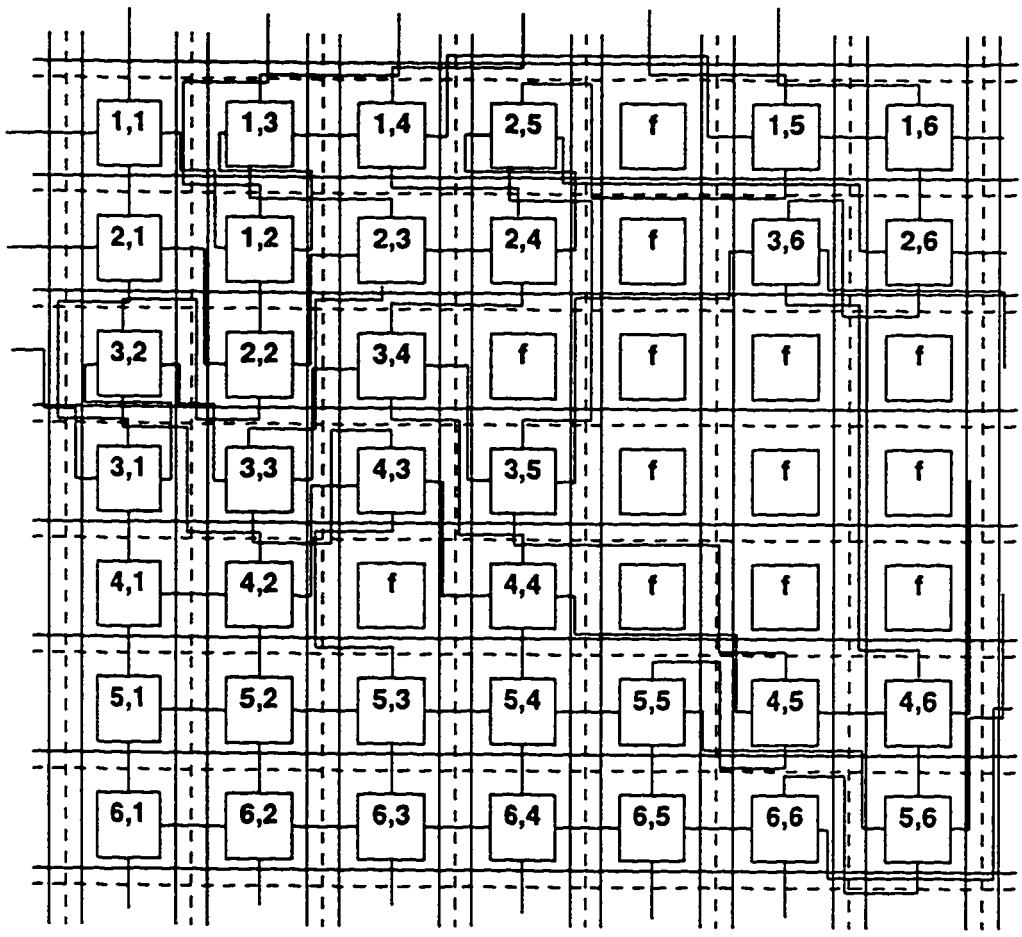


Figure 5.11: The final solution obtained by using greedy algorithm on 3 × 2 interconnection resources.

Initial solutions can be formed in two ways, pattern based or knowledge based. In pattern based initial solution a single initial solution is used for all types of fault pattern. While simple patterns like plus pattern can be placed using procedures of constant complexity, other patterns may require the use of the framework.

For knowledge based initial solution a new initial solution has to be formed for each fault pattern. This can be done by using bipartite matching which has a complexity of cubic order i.e. $O((M \times N)^3)$. Simple matching procedure used in [21] has complexity of quadratic order.

$$Complexity\ of\ Knowledge-based\ Initial\ Solution = O((M+N)^2) \qquad (5.1)$$

Reconfiguration with the framework is performed using the three layers. Each step taken in layer 3 makes use of utilities used in layer 2 which inturn makes use of rule checking procedures provided in layer 1. Therefore, the overall complexity of reconfiguration is the product of the complexity of the three layers.

If we assume that the size of the array is $M \times N$ and there is one spare row and one spare column in the array then there are $S = M + N - 1$ spare elements in the array. In the worst case the array can be reconfigured if there are S faulty elements in the physical array.

In order to evaluate the complexity of the framework we begin by evaluating the complexity of the third layer, and then the second layer and then the core. In the third layer there are two techniques, greedy algorithm and incremental distance

algorithm. In greedy algorithm if there are S faulty elements and S spares then for each impaired element we may have to search all spares. Therefore, greedy algorithm has complexity

$$Complexity\ of\ Greedy\ Algorithm = O(S^2) \tag{5.2}$$

In incremental distance algorithm, for the worst case we may require $M+N-1 = S$ passes in which in each pass there are S impaired elements and for each impaired element we have to search S spare elements. Therefore, the worst case complexity of the incremental distance algorithm is

$$Complexity\ of\ Incremental\ Algorithm = O(S^3) \tag{5.3}$$

Once a spare element is found for an impaired elements then the utilities used in layer 2 have to be used to find a valid path for reconfiguration. The number of direct paths between any two elements $(x1, y1)$ and $(x2, y2)$ in the processor array are $\binom{(x1-x2)}{(x1-x2)+(y1-y2)}$. In the worst case the two elements are placed at the two corners elements on the diagonal of the square array. The distance between them is S and $(x1 - x2) = M \approx S/2$. Therefore, complexity of complete tree search is

$$Complexity\ of\ tree\ search = O(\binom{S/2}{S}) = O(S)^{S/2} \tag{5.4}$$

This of exponential complexity. As opposed to this in the four path search we always search four paths only. The complexity of four path search is constant.

$$Complexity\ of\ four\ paths = O(constant) \tag{5.5}$$

Finally, for validating a move we have to check rules. Each element along the path has to be checked. In the worst case there are S elements along the path. For each element we require computation of constant complexity for any type of interconnection resources. Therefore, complexity of the the core is of the order

$$Complexity\ of\ core = O(S) \tag{5.6}$$

If we use knowledge based initial solution, incremental distance algorithm and tree search then complexity of reconfiguration from above equation is

$$Complexity\ of\ framework = O(S^2 + S^3 \times S^{S/2} \times S) = O(S^{9S/2}) \tag{5.7}$$

Therefore, when tree search is used complexity of framework is exponential.
The complexity is greatly reduced if four-path search is used. Complexity of framework for the same techniques but using four-path search is

$$Complexity\ of\ framework = O(S^2 + S^3 \times C \times S) = O(S^4) \tag{5.8}$$

where C is a constant.
Though the equation indicates overall complexity of polynomial order of degree 4, in reality the complexity is well within cubic order of complexity.

5.7 Concluding Remarks

As the size of VLSI circuits is getting larger there is a need to implement large size chips. Making large size chips on WSI is impractical without adding redundancy.

However, as the size of the array grows, yield falls rapidly. Therefore large area circuits have to be made with optimal amount of redundancy.

In processor arrays redundancy can be in two forms : spare element redundancy and switch and track redundancy. The optimal combination of redundant spare elements and interconnection resources in the form of switches and tracks has to be carefully chosen to make fabrication cost-effective.

In this chapter a framework for yield enhancement of processor arrays has been proposed. The framework is implemented in three layers and allows us to make estimation of survivability of an array. Layer 1 is the core and provides a platform to evaluate the feasibility of the current placement of the logical cells for different amounts of interconnection resources. The second layer provides utilities to search spares which are used by the reconfiguration techniques proposed in layer 3. Two techniques have been proposed for reconfiguration in processor arrays. Both techniques start reconfiguration from a completely placed initial solution.

Measures for obtaining good pattern based spare distribution have been proposed. Using these measures initial solutions are made. Method of formation of initial solution using incremental placement techniques is also presented. A range of good initial solutions can be obtained by using the two approaches.

Using the initial solutions we obtain the final solutions using reconfiguration techniques of layer 3. The successful reconfiguration rate determines survivability.

Chapter 6

Working with the Framework

This chapter provides the readers with the details of reconfiguration with different types of interconnection resources. We consider an example which explains the difference between the different reconfiguration approaches used and the difference in the final outcome when different types of interconnection mechanisms are used.

In Section 6.1 we present an example in which we reconfigure the array using greedy approach for 3×3 and then with 3×2 interconnection resources for the same fault pattern. In Section 6.2 reconfiguration is repeated for the same fault pattern for both 3×3 and 3×2 interconnection resources using incremental distance algorithm. Using the same fault pattern we also show how final configuration can be obtained using the knowledge based spare distribution in Section 6.3. In Section 6.4 we present some concluding remarks.

6.1 Reconfiguration using Greedy Algorithm

In this section we present an example in which we reconfigure a faulty array of size 7 × 7 using the greedy algorithm. We start reconfiguration from a plus pattern based initial solution. The plus pattern based initial solution is valid for both 3 × 3 and 3 × 2 interconnection resources. There is one row and one column of spare elements. Therefore, the final array is of size 6 × 6.

In Figure 6.1 the physical elements with logical cells (0,0) are spare elements. Location of faulty elements are indicated with dashed boxes. A physical element which is faulty and has a logical cell assigned to it is called an impaired element. Elements which are spare and faulty are bad elements.

To reconfigure impaired elements we make a sequence of moves from the initial

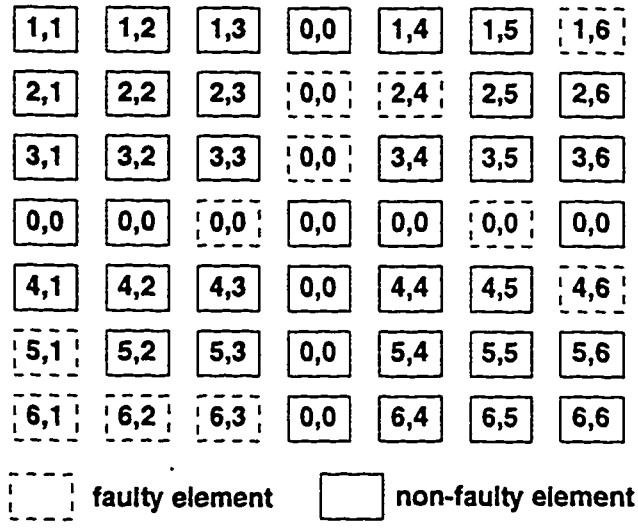


Figure 6.1: Plus pattern initial solution for 7 × 7 physical array and 6 × 6 logical array.

solution. After each move the rules for interconnection resources have to be checked to prevent congestion in channels. Therefore, initially we have to decide the type of interconnection resources that will be used. In this example we illustrate the use of 3×3 and 3×2 interconnection resources.

For 3×3 interconnection resources the first four steps are shown in Figure 6.2(a).

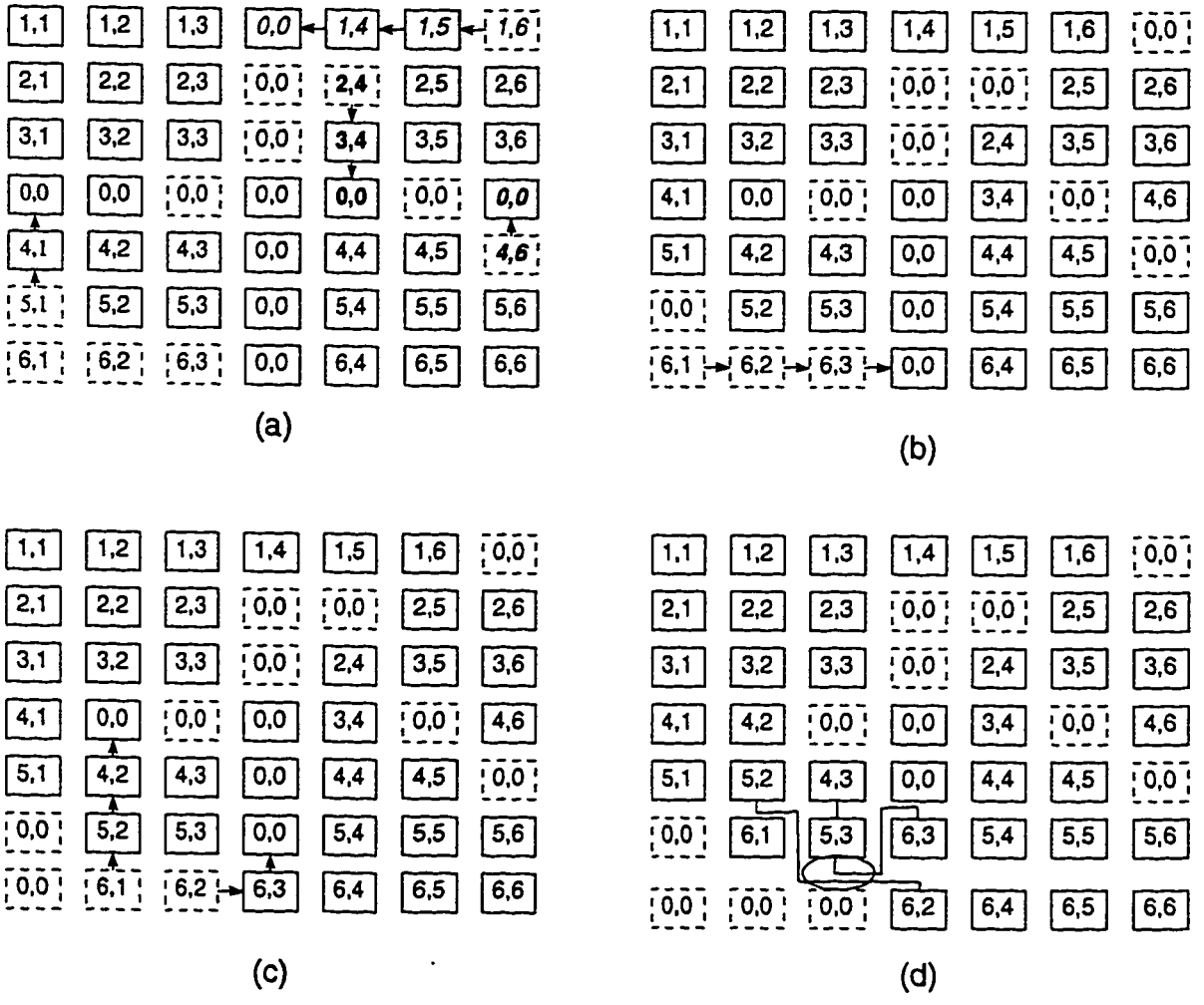


Figure 6.2: Reconfiguration using greedy algorithm with 3×3 interconnection resources.

When the array is scanned the first impaired element met is $(1,7)_P$. The spare elements are searched in a clockwise direction starting from the left horizontal (i.e. $(1,7-d)_P$, where d is distance). There are two spares at distance 3 for the impaired element $(1,4)_P$ and $(4,7)_P$. Since, $(1,4)_P$ is met first (while scanning), we move the logical cell on the impaired element to the spare element. The move does not cause any conflicts for 3×3 interconnection resources. Therefore, $(1,7)_P$ is moved to location $(1,4)_P$ as indicated in Figure 6.2(a). The next impaired element in sequence is $(2,5)_P$. The impaired element is successfully moved to location $(4,5)_P$. Next impaired element $(5,7)_P$ is moved to location $(4,7)_P$. In the fourth step impaired element $(6,1)_P$ is moved to $(4,1)_P$. The four steps are shown in Figure 6.2(a). It should be noted that for all the four steps we check rules for every move.

In the fifth step $(7,1)_P$ is assigned to $(7,4)_P$. There are two impaired elements in the only direct path between the two elements. As a rule a logical cell is moved to an element which has a logical cell (impaired or functional element) or a spare element. The move does not cause a conflict, therefore, it is taken as shown in Figure 6.2(b). This move causes a change in logical cells on the impaired elements. The last two steps are shown in Figure 6.2(c). In the final configuration shown in Figure 6.2(d) we see that the channel between $(6,3)_P$ and $(7,3)_P$ requires 3×3 interconnection resources. This configuration is therefore not valid for 3×2 interconnection resources.

For 3×2 interconnection resources the first eight steps are shown in Figure

6.3(a) to Figure 6.3(c). These steps are the same as the eight steps taken for 3×3 interconnection resources. However, the ninth step for 3×3 interconnection resources leads to a resource conflict with 3×2 interconnection resources. Then the next path is checked and the final configuration is obtained as shown in Figure 6.3(d).

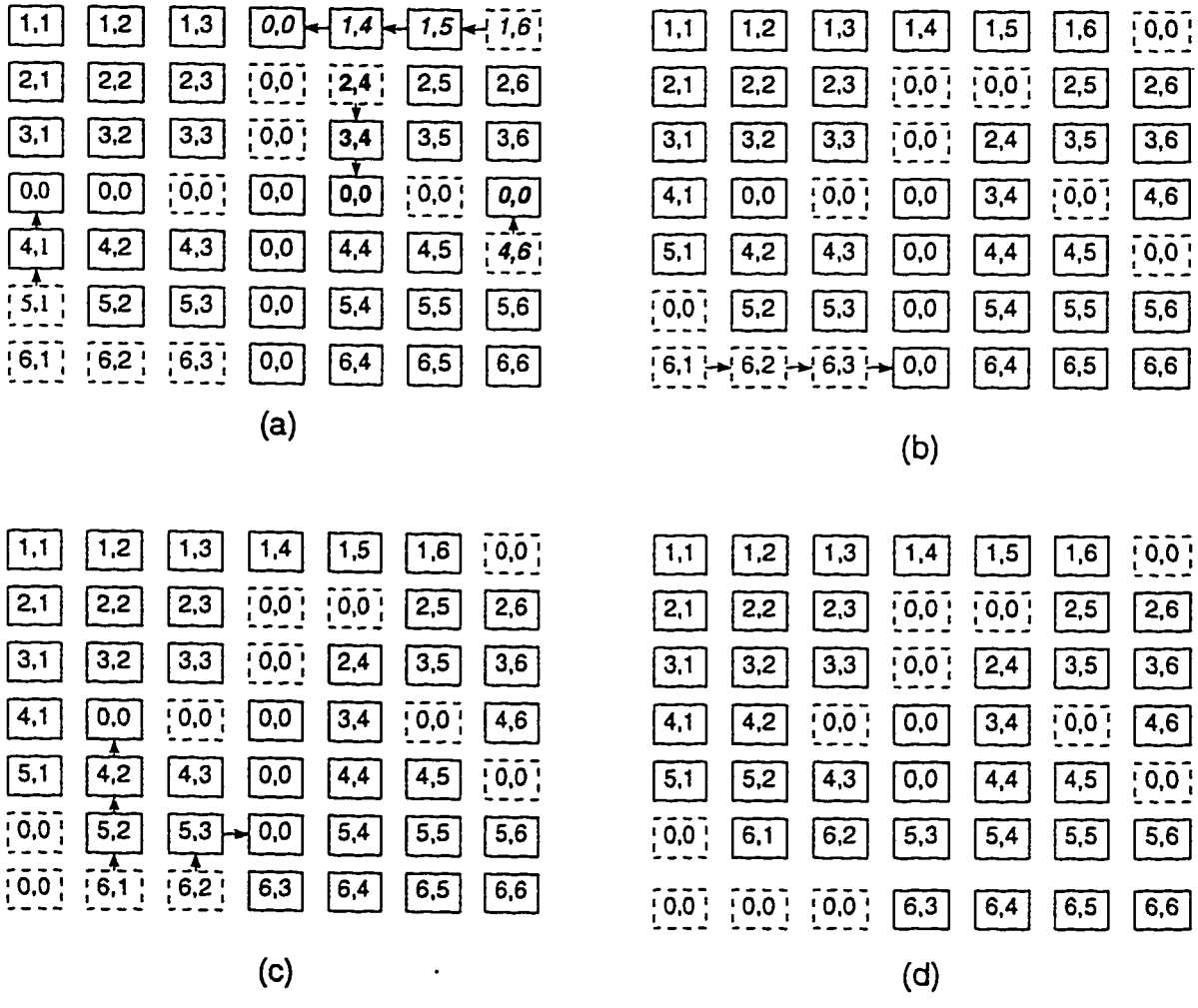


Figure 6.3: Reconfiguration using greedy algorithm with 3×2 interconnection resources.

6.2 Reconfiguration using Incremental Distance Algorithm

In incremental distance algorithm, reconfiguration of impaired elements is carried out in passes. In each pass a few impaired elements are reconfigured. In each pass the manhattan distance at which spares for impaired elements have to be searched is initialized. In a pass the array is scanned in raster pattern and when an impaired element is met, spare elements at the initialized manhattan distance are searched. If a spare element is found and the move from the impaired element to the spare element does not cause a conflict then, the move is made. If no spare element exists at the specified manhattan distance then that impaired element is reconfigured in the subsequent passes.

In this section we illustrate the reconfiguration of a 7 × 7 faulty PA with 3 × 3 and 3 × 2 interconnection resources. We have used the same fault pattern as is used in the previous section.

First, we show the reconfiguration of the faulty PA with 3 × 3 interconnection resources. In the first pass with manhattan distance 1, two impaired elements are reconfigured. The two elements are $(5,7)_P$ assigned to $(4,7)_P$ and $(7,3)_P$ assigned to $(7,4)_P$. This is shown in Figure 6.4(a). In pass 2 two more impaired elements are reconfigured. The two elements are $(2,5)_P$ assigned to $(1,4)_P$ and $(6,1)_P$ assigned to $(4,1)_P$ as shown in Figure 6.4(b). To move $(2,4)_L$ we cannot move it to the

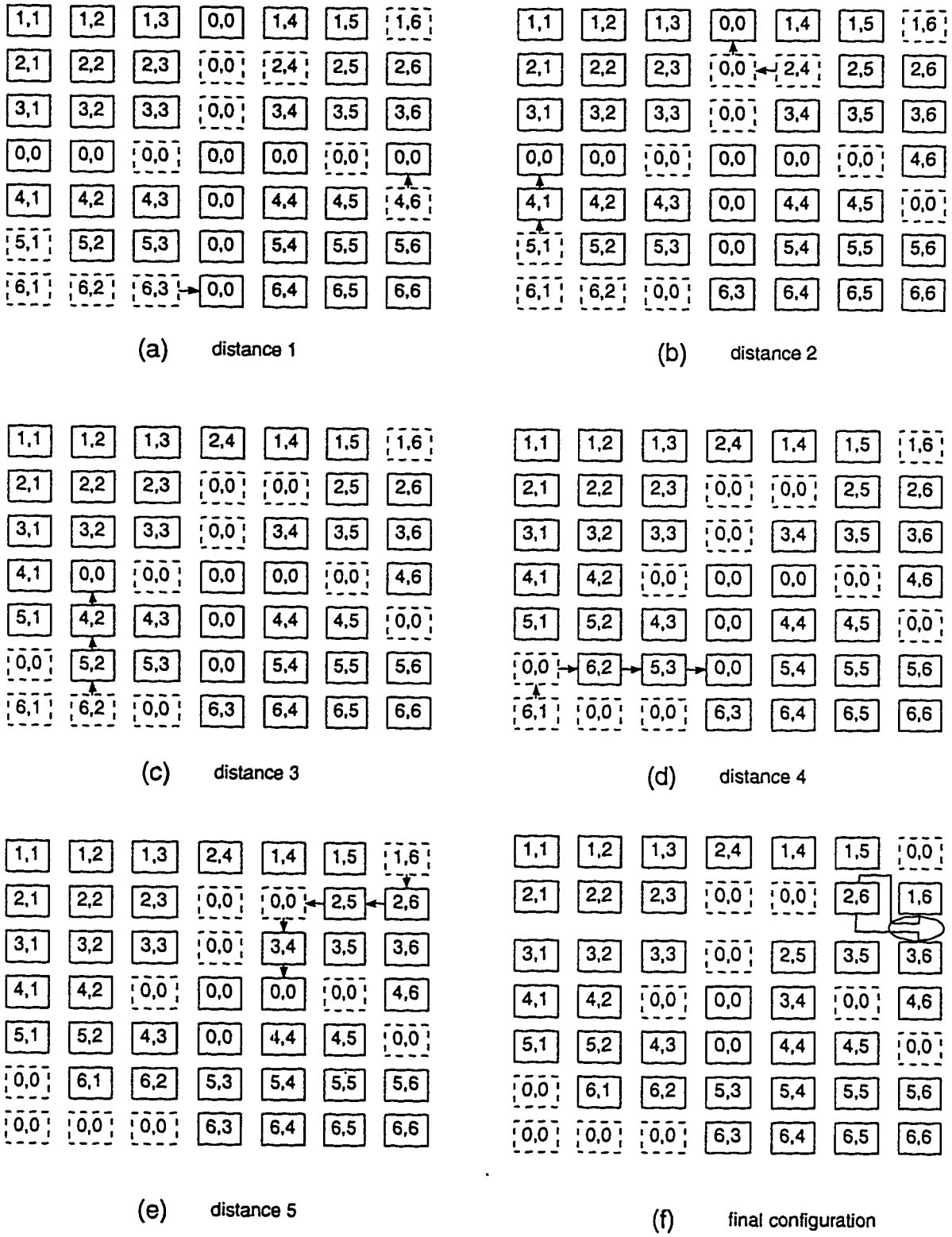


Figure 6.4: Reconfiguration using incremental distance algorithm with 3 × 3 interconnection resources.

adjacent element in the path as it is a bad element. Therefore, it is moved to the next element along the path i.e. $(1,4)_P$.

In the subsequent passes one impaired element is reconfigured in each pass as shown in Figure 6.4(c) to Figure 6.4(e). A bad element along the path is not used in the move as shown in Figure 6.4(d) and Figure 6.4(e). The final configuration as shown in Figure 6.4(f) has a duplicate assignment due to which there will be congestion for 3×2 interconnection resources in the channel shown with a spot light.

Reconfiguration of the faulty PA with 3×2 interconnection resources is made in five passes. In the first four passes the same steps are taken. However, in the fifth pass with manhattan distance 5 the path shown in Figure 6.4(e) leads to violation of duplicate assignment rules for 3×2 interconnection resources. Therefore, a new path is tried as shown in Figure 6.5(e). The new path leads to a final configuration as shown in Figure 6.5(f). The final configuration is valid for 3×2 interconnection resources.

Reconfiguration of a large sized processor array i.e. 15×15 array with 225 PEs is explained in Appendix B. For larger size PAs most of the moves are with manhattan diatance greater than 4.

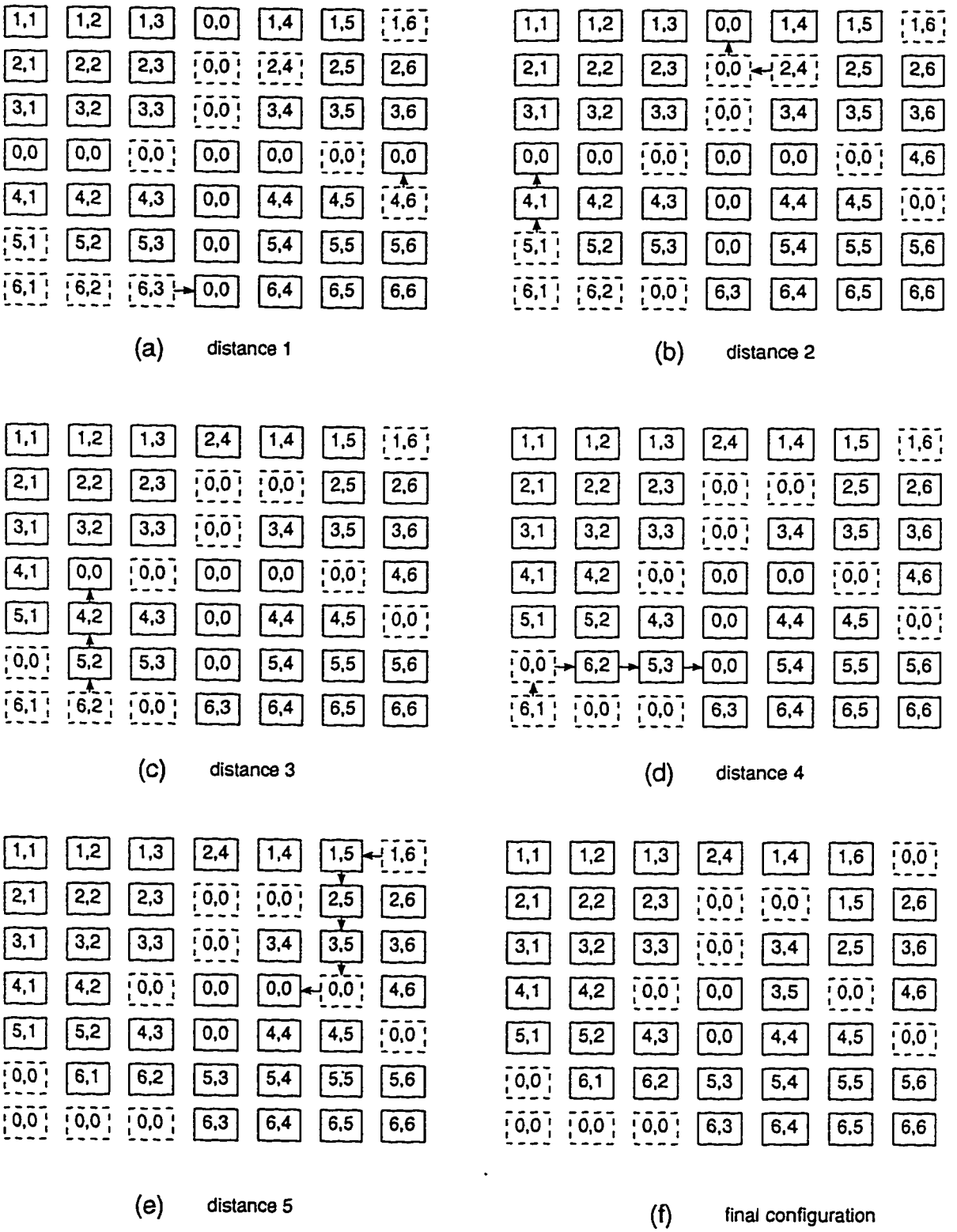


Figure 6.5: Reconfiguration using incremental distance algorithm with 3 × 2 interconnection resources.

6.3 Reconfiguration using Knowledge Based Spare Distribution

Knowledge-based reconfiguration (also called incremental placement) is based on variable domain approach. In the variable domain approach the array is first divided into overlapping row and column subranges. The region which is an intersection of a row and a column subrange defines the domain of a logical cell [21]. Each logical cell can be assigned to a set of physical elements. Once all the domains of logical cells are defined then a matching is made between the logical cells and physical elements. The resulting matching is translated into a possible configuration. The final configuration obtained using variable domain approach is valid only for 3×3 interconnection resources.

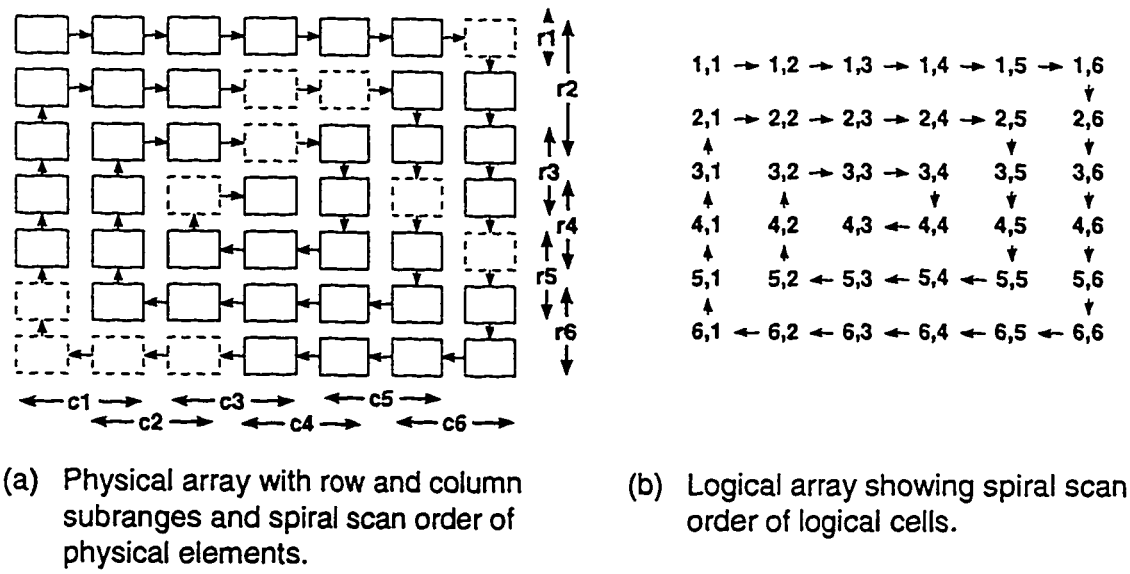
In this section we illustrate with an example the formation of a configuration using incremental placement technique. The size of the physical array is 7×7 and size of the logical array is 6×6. For convenience we use the same fault pattern used in the previous sections. The faulty PA with logical row and column ranges is shown in Figure 6.6.

A logical row range is the range of physical rows to which cells of a logical row can be assigned safely without causing conflicts in channels with 3×3 interconnection resources. The array is divided into as many row ranges as there are logical rows. To maximize the range of a logical row we stretch the ranges to the maximum limits

allowable with 3×3 interconnection resources. The procedure for forming the logical ranges is described in Chapter 4.

The overlapped region of a logical row range and a logical column range determines the domain of a logical cell. The domain of a logical cell determines the edges between logical cells and physical elements.

Logical cells are matched to physical elements based on the procedure suggested in [21]. The physical elements and logical cells are scanned in a spiral pattern as indicated in Figure 6.6.



(a) Physical array with row and column subranges and spiral scan order of physical elements.

(b) Logical array showing spiral scan order of logical cells.

Figure 6.6: Row column ranges of a faulty PA. Spiral scan of physical elements and logical cells is also shown.

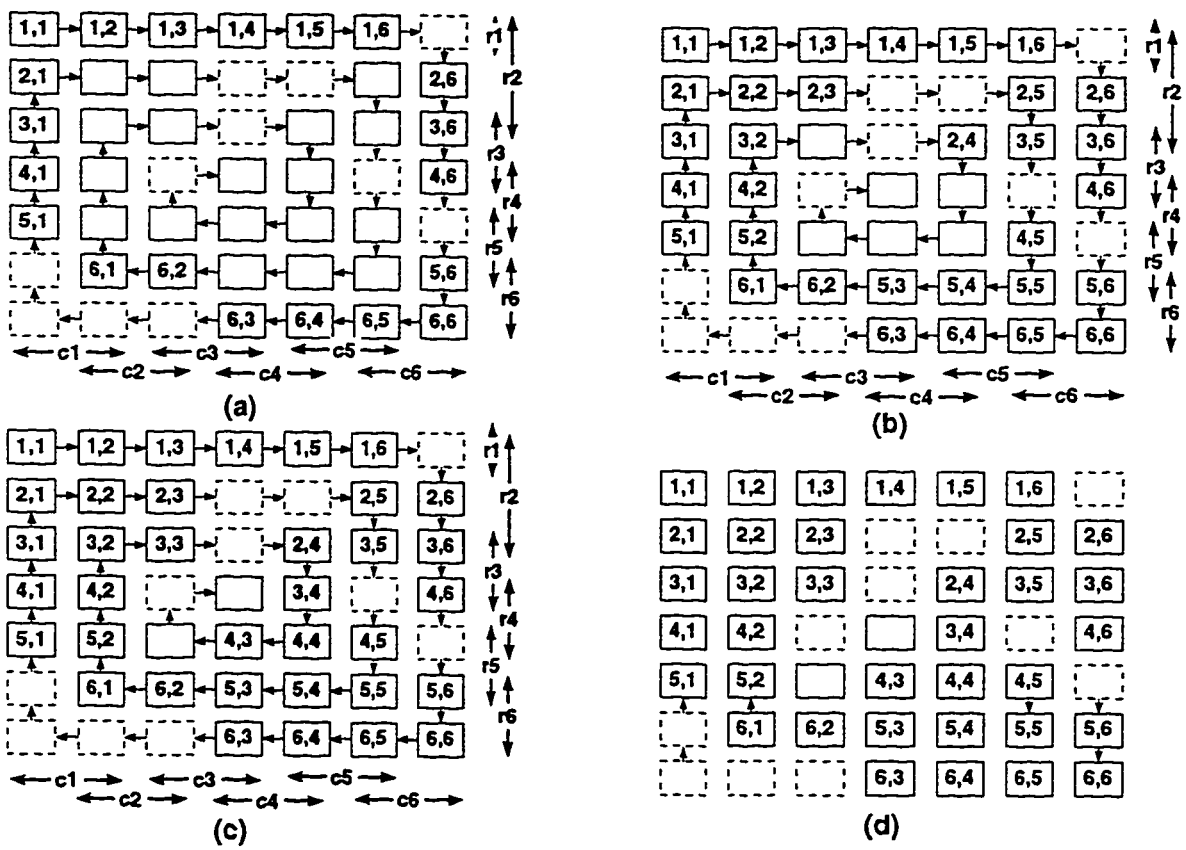


Figure 6.7: Incremental placement of logical cells on the physical array avoiding the faulty elements (knowledge based reconfiguration).

The incremental placement of logical cells on the physical array is illustrated in four steps in Figure 6.7. We try to assign each logical cell to any one of the physical elements on the array in the spiral scan order. To assign logical cell $(6,2)_L$ we check for a free non-faulty physical element on the array in the order $(1,1)_P$, $\cdots$, $(1,7)_P$, $(2,7)_P$, $\cdots$, $(7,7)_P$, $(6,7)_P$, $\cdots$, $(7,1)_P$, $(6,1)_P$, $\cdots$, $(2,1)_P$, $(2,2)_P$, $\cdots$, $(2,6)_P$, $(3,6)_P$, $\cdots$, $(6,6)_P$, $(5,6)_P$, $\cdots$, $(3,6)_P$. The first non-faulty element to which the logical cell can be assigned is met. Therefore, $(6,2)_L$ is assigned to $(3,6)_P$. Similarly, the other logical cells are assigned as shown in Figure 6.7(a) to Figure 6.7(c). The final configuration is shown in Figure 6.7(d). The final configuration is valid for both 3×3 and 3×2 interconnection resources.

6.4 Concluding Remarks

In this chapter we have illustrated with examples the reconfiguration of faulty PAs using different reconfiguration techniques using 3×3 and 3×2 interconnection resources. We see that the final configurations obtained using rules for 3×2 interconnection resources are different from those obtained with 3×3 interconnection resources. The final configurations obtained for the given fault pattern for different reconfiguration techniques with 3×3 and 3×2 interconnection resources are different. This indicates that multiple final configurations exist for any fault patterns

and a suitable final configuration for a given type of interconnection resources can be obtained by using rules for that type of interconnection resources.

Chapter 7

Results and Discussion

The framework for yield enhancement of processor arrays has been modeled in three-layers. The three layer model allows us to make different kinds of experimental estimations. In addendum to the three layer model, the framework also incorporates two other modules, one for generating spare patterns on the array and the other for making fault distributions based on the clustering parameter. Use of these modules is not binding as they are not part of the framework. Using all the modules of the framework, different types of experiments can be made.

The core of the framework supports four different types of switch and track interconnections. The four interconnection fabrics are 3×3, 3×2, 2×3 and 2×2. As the amount of interconnection resources are reduced, reconfiguration of a faulty arrays becomes more difficult.

To compensate for the loss of survivability due to lesser interconnection resources,

we propose to make a more extensive search of the sample space of the FFA. In order to make this possible two utilities are provided in the second layer, which allow different levels of search for a suitable spare to reconfigure an impaired element.

The third layer provides reconfiguration techniques for faulty arrays. Two reconfiguration techniques are proposed. They are the greedy approach and the incremental distance approach.

Apart from the three layer model of the framework, the other modules that are incorporated to make the framework more versatile are the modules for initial spare distribution and the fault pattern distribution. Using the infrastructure provided by the framework different types of experiments can be conducted to make predictions about yield of processor arrays. As an application we study the effect of degradation in performance of survivability for larger sizes of the array.

The rest of the chapter is organized as follows. In section 7.1 we present comparison of different techniques provided in each layer of the framework. Comparison of results of reconfiguration from two different types of formation of initial solutions are given in section 7.2. Comparison of results of survivability of incremental placement approach of semi-clustered and heavily-clustered fault patterns is given in section 7.3. In section 7.4 we present results obtained for different sizes of processor arrays. Discussion based on these results is presented in section 7.5 and the conclusions of this chapter are given in section 7.6.

7.1 Comparison of Techniques in Different Layers

The first layer of the framework is the core. It provides the functionality of validating placement of logical cells on the processor arrays for different switched bus interconnection resources. The interconnection architectures that are supported by this architecture are 3×3, 3×2, 2×3 and 2×2. As an example, we present comparison of survivability obtained for 3×3 and 3×2 switch bus interconnections using incremental placement in section 7.1.1

The second layer of the framework is built on the core. It provides the utilities to search for suitable spares in the array for reassignment. There are two utilities in this layer. They provide different degrees of search in the array. The first utility is based on depth first search of the tree formed within the bounding box of the spare and the impaired element (tree). The second utility searches atmost four paths of the complete tree with less than three bends. Comparison of results of the tree and four paths using the incremental distance algorithm are presented in section 7.1.2.

The third layer is built on the first and second layer of the framework. Using this layer, different reconfiguration approaches can be applied. We have proposed two techniques for reconfiguration of processor arrays. The two techniques are the greedy approach and incremental distance approach. In the greedy approach we try to reconfigure an impaired element greedily. On the other hand in the incremental

distance approach reconfiguration is carried out in passes of incremental distance on the array. Comparison of results of the two reconfiguration techniques are given in section 7.1.3.

7.1.1 Comparison of Interconnection Resources

The most widely used switched bus interconnection resources are 3×3 and 3×2. We compare their performance on the framework using the incremental placement approach. The initial solution (complete) is obtained using the knowledge based spare distribution approach for both types of interconnections. The clustering parameter is 1.0. The fault patterns of the PEs on the processor array is semi-clustered.

Complete results of the comparison are shown in Figure 7.1. For convenience some of the results are tabulated in Table 7.1. It can be seen from the table that if the number of faulty elements is equal to the number of spares (spare demand $\rho = 1$) then for semi-clustered fault patterns survivability falls below 0.9. If the number of faults are less than number of spares (spare demand $\rho < 1$) then survivability is fairly high (above 0.9).

7.1.2 Comparison of Search Techniques

The second layer performs the function of searching suitable spares for an impaired element in the array. Since only direct paths are considered the total number of

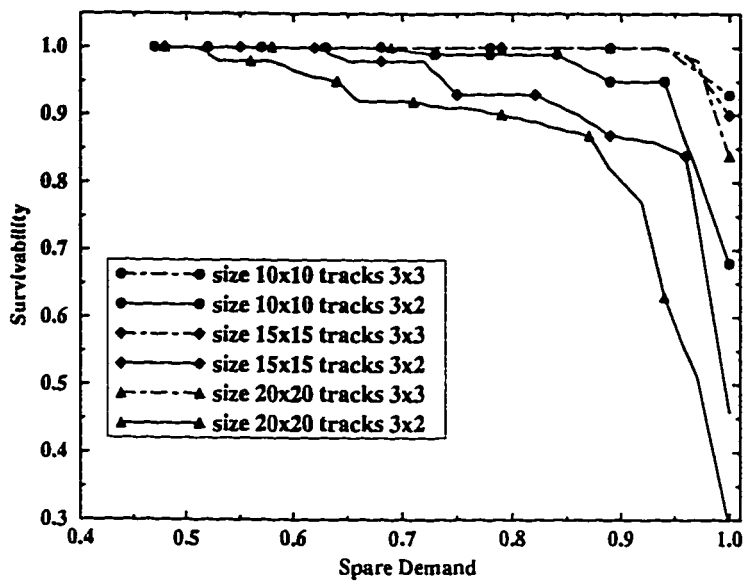


Figure 7.1: Comparison of survivability of 3 × 3 with that of 3 × 2 switch bus interconnection for different sizes of the array.

Spare demand	10x10		15x15		20x20	
	3x3	3x2	3x3	3x2	3x3	3x2
0.50	100	100	100	100	100	100
0.75	100	99.3	100	94	100	91
1.00	93	68	90	46	84	29

Table 7.1: A comparison of array survivability of 3 × 3 and 3 × 2 switch bus interconnection for different sizes of the array.

paths are equal to the number of paths in a tree.

If m is the difference in the horizontal coordinates of the impaired element and

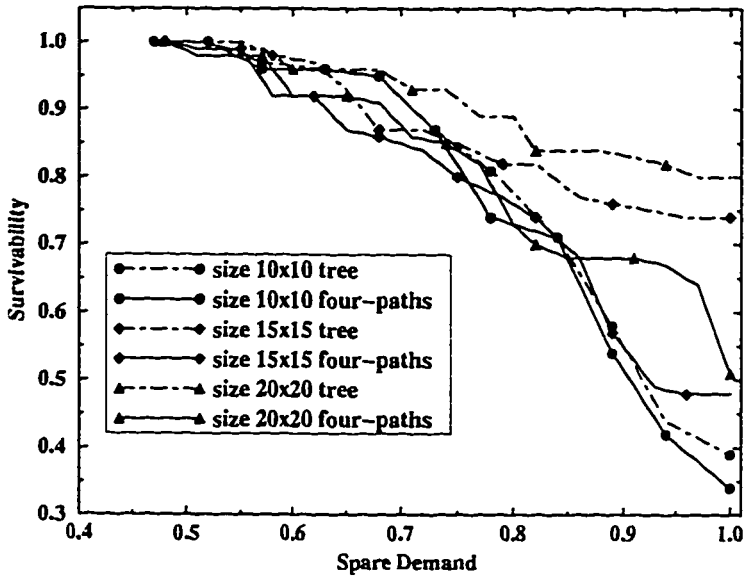


Figure 7.2: A comparison of search techniques. The two types are tree search and four-paths search.

the spare element and n is the difference in the vertical coordinates, then there are $\binom{m+n}{n}$ paths threads from the spare to the impaired element. If all these paths are searched then the complexity of the search is $O(m^n)$, which is exponential. To reduce the complexity of the search for a suitable spare we have a utility which searches only four paths out of the $\binom{m+n}{n}$. Therefore, the complexity of search for a suitable spare is constant.

Comparison of results when the two types of search are used for incremental distance algorithm on 3 x 3 interconnection resources, using plus pattern based spare

distribution, and clustering parameter $\alpha = 1.0$ are shown in Figure 7.2. Results for 50%, 75% and 100% faulty to spare percentage are tabulated in Table 7.2.

From Table 7.2 we see that for very high values of faulty to spare element per-

Spare demand	10x10		15x15		18x18	
	tree	four-path	tree	four-path	tree	four-path
0.50	100	100	100	100	100	100
0.75	84	80	85	80	93	85
1.00	39	34	74	48	80	51

Table 7.2: A comparison of array survivability of tree and four-path search of suitable spare for different sizes of the array.

centage there is tangible difference in survivability of tree search over the four-path search. If number of spare elements are more than number of faulty elements then both techniques have similar figures of survivability.

The results were expected because four-path search performs a limited search of the available direct paths.

7.1.3 Comparison of Proposed Reconfiguration Techniques

A range of reconfiguration techniques can be applied to reconfigure faulty processor arrays. Two reconfiguration schemes were proposed to exploit the utilities provided by the first and second layer of the framework. These are the greedy approach to reconfiguration and incremental distance approach.

In the greedy approach when an impaired element is found, we look up for an-

other impaired element to reconfigure only when the current impaired element is successfully reconfigured. The order of search of impaired elements is from left top to right bottom in raster pattern i.e. in the order (row_1,column_1), (row_1,column_2), ... , (row_1,column_N), (row_2,column_1), ... , (row_M,column_N), where the array size is $M \times N$. If there are p spare elements due to which atmost p faulty elements can be reconfigured, then in the worst case for each of the p faulty elements p spares have to be searched. Therefore, complexity of greedy approach is $O(p^2)$ i.e. quadratic.

In incremental distance algorithm reconfiguration is performed in steps of incremental distance d. When an impaired element is found we try to reconfigure the impaired element only if we find a spare element at distance d from the impaired element. If a suitable spare is not found then the current impaired element is left and the next impaired element in the array is tried for reconfiguration. The order of search of impaired elements is performed in raster pattern.

If there are p spare elements to which p impaired elements have to be reconfigured then we perform reconfiguration in passes for $d = 1$ to $d = p$ (for one spare row and one spare column, maximum distance is $M + N - 1$). Therefore, we make p passes for reconfiguration. Therefore, complexity of incremental distance algorithm is $O(p^3)$ i.e. cubic.

Comparison of the two types of reconfiguration approaches was done using plus pattern initial spare distribution and four-path spare search on 3×3 switch bus interconnection. Results of the two reconfiguration approaches are shown in Figure

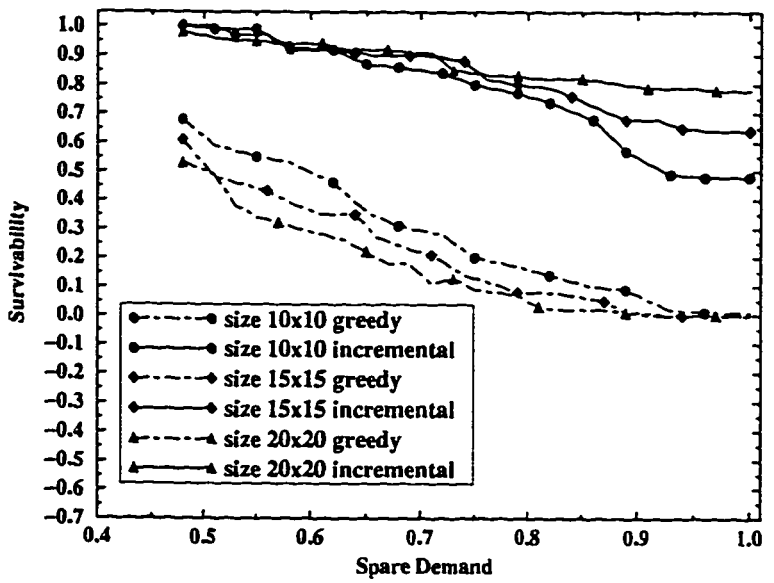


Figure 7.3: Comparison of survivability of the greedy reconfiguration approach with that of incremental distance approach.

7.3. Comparison of two reconfiguration approaches for spare demand 0.5, 0.75 and 1.00 is shown in Table 7.3.

From Table 7.3 it can be seen that performance of incremental distance algorithm

Spare demand	10x10		15x15		20x20	
	greedy	incremental distance	greedy	incremental distance	greedy	incremental distance
0.50	80	100	68	100	61	100
0.75	41	80	20	80	13	85
1.00	16	34	1	48	0	51

Table 7.3: A comparison of array survivability of greedy algorithm and incremental distance algorithm.

is much better than the greedy algorithm especially when the spare demand approaches 1. We see a similar trend as the size of the array increases. For large size arrays survivability of greedy algorithm is very low even for spare demand 0.5. Greedy approach does not use the resources efficiently. For an impaired element it seeks a spare element greedily inconsiderate of how the other impaired elements would be configured. On the other hand in the incremental distance algorithm if an impaired element cannot be moved within a specified distance we move ahead and try to reconfigure other impaired elements. The more difficult to reconfigure impaired elements are configured in the end thereby retaining the structure of the logical array till the end.

7.2 Changes in Formation of Initial Solutions

In order to give the user the flexibility of applying different reconfiguration approaches to the framework, two spare distribution and initial solution formation techniques have been incorporated in the framework. The two techniques are pattern based spare distribution and knowledge based spare distribution. Pattern based spare distribution is a static spare distribution in which spare elements form a pattern. Initial solutions obtained for an array is used for all fault distributions. The simplest pattern with low accessibility is the plus pattern. Knowledge based spare distribution forms an initial solution by substituting the faulty elements.

While pattern based spare distribution is static and does not require computation to form the complete initial solution, knowledge based spare distribution requires bipartite matching and subsequently placement of unassigned logical cells. Therefore, its complexity is $O(L^3)$, where $L \times L$ is the size of the logical array.

Comparison of the plus pattern based spare distribution and knowledge based spare distribution is shown in Figure 7.4. The switch bus interconnection used for this comparison is 3×3, the spare search technique used is four-path technique and the reconfiguration algorithm used to reconfigure the complete placement is incremental distance algorithm.

Comparison of the results in tabular form for two types of initial solution formation is given in Table 7.4.

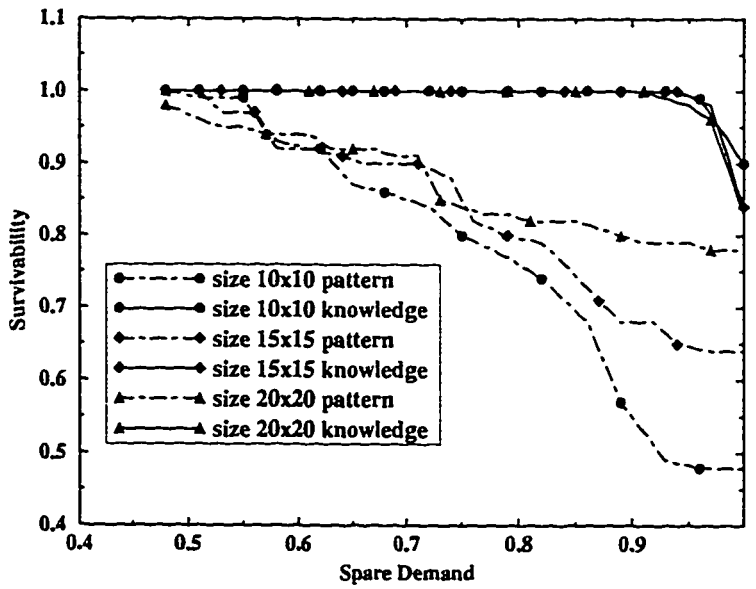


Figure 7.4: Comparison of survivability using plus pattern based initial solution and knowledge-based initial solution.

From Table 7.4 it can be seen that as spare demand increases (approaches 1)

Spare demand	15x15		20x20		25x25	
	pattern based	knowledge based	pattern based	knowledge based	pattern based	knowledge based
0.50	100	100	100	100	97	100
0.75	80	100	85	100	84	100
1.00	48	90	64	84	78	83

Table 7.4: Comparison of array survivability using plus pattern based initial solution and knowledge-based initial solution.

knowledge based spare distribution performs better than pattern based spare distribution. This is expected because knowledge-based spare distribution is adaptive. Formation of the initial solution in knowledge-based spare distribution is done by avoiding faulty elements and matching the non-faulty elements. This makes a very efficient placement of logical cells.

7.3 Changes in Clustering Parameters

The most important factor in making realistic estimates of survivability is to distribute faults in the array based on different clustering tendencies. At a time a manufactured IC lot is expected to exhibit fault clusters of a certain type depending on various manufacturing parameters. Fault prediction models use the clustering parameter to model different fault clusters. The framework has a module which distributes faults based on the large-area fault model. Using the fault distributor,

we have conducted simulation of semi-clustered ($\alpha = 1.0$) and heavily-clustered ($\alpha = 0.3$) fault distributions.

The other parameters used are 3 x 3 size of interconnection resources, knowledge based spare distribution, four-path spare searching technique and greedy reconfiguration algorithm. Comparison of results of the two cluster types are shown in Figure 7.5.

Comparison of the results in tabular form for heavily clustered ($\alpha = 0.3$) and

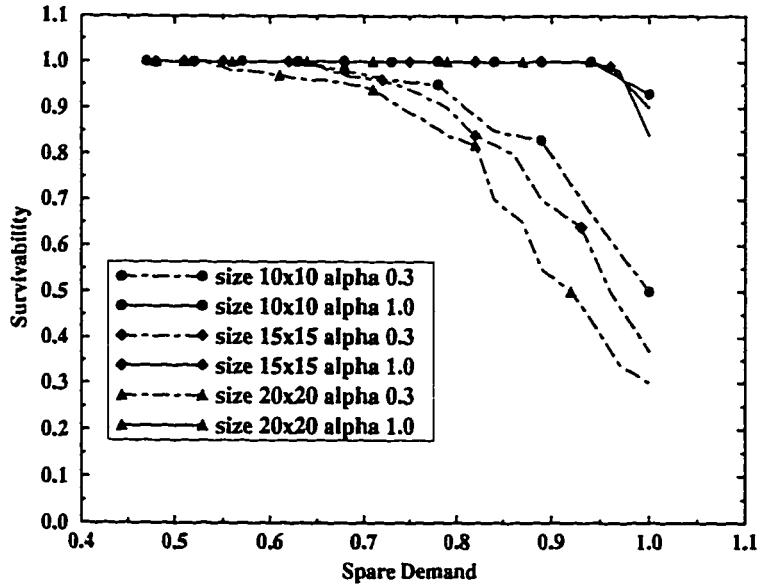


Figure 7.5: Comparison of survivability based on clustering parameter.

semi-clustered ($\alpha = 1.0$) faults are given in Table 7.5.

From Table 7.5 it can be seen that when faults are heavily clustered then survivability falls rapidly with increasing spare demand and size of the array. The reason

Spare demand	10x10		15x15		20x20	
	$\alpha = 0.3$	$\alpha = 1.0$	$\alpha = 0.3$	$\alpha = 1.0$	$\alpha = 0.3$	$\alpha = 1.0$
0.50	100	100	100	100	100	100
0.75	96	100	94	100	90	100
1.00	50	93	37	90	3	84

Table 7.5: Comparison of survivability based on clustering parameter.

is that interconnection resources in the region of heavy-clusters cannot be put to use. If the faults are semi-clustered then almost all the interconnection resources can be put to use.

7.4 Variations with the Size of the Array

Most of the reconfiguration approaches presented so far experience a degradation of performance as the size of the processor array increases. Therefore, due to considerations of feasibility of implementation, size of square arrays larger than 20×20 are not reported in most of the literature [13] [4].

Using the different reconfiguration approaches we conducted experiments with the size of the array. Simulations were conducted for different reconfiguration approaches implemented on the framework. These are the incremental placement technique using knowledge based spare distribution (remainder logical cells placed using greedy algorithm), greedy algorithm using plus pattern spare distribution and incremental

distance algorithm using plus pattern spare distribution. Interconnection resources used in all the three reconfiguration approaches is 3×3. Clustering parameter is $\alpha = 1.0$ and four-paths spare searching technique.

Results of the reconfiguration capabilities of different techniques for varying sizes of the arrays are shown in Figure 7.6, Figure 7.7 and Figure 7.8.

Results in tabular form for incremental placement approach are given in Table 7.6.

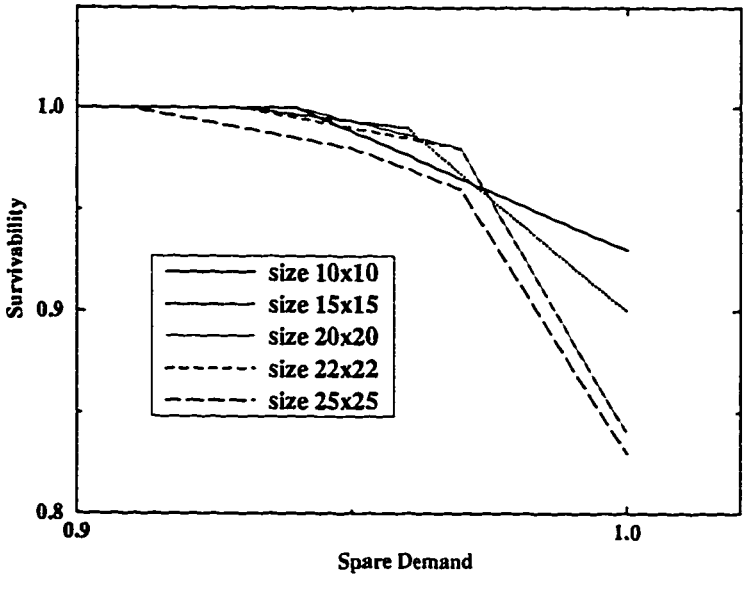


Figure 7.6: Variation in survivability of the array for increasing size of the array for incremental placement using knowledge based spare distribution.

From Table 7.6 it can be seen that survivability of the array falls with increasing size of the array for higher values of spare demand.

Results in tabular form for incremental placement approach are given in Table 7.7.

From Table 7.7 it can be seen that survivability of the array falls with sharply with

Spare demand	10x10	15x15	20x20	22x22	25x25
0.50	100	100	100	100	100
0.75	100	100	100	100	100
1.00	93	90	84	83	80

Table 7.6: Variation in survivability of the array for increasing size of the array for incremental placement using knowledge based spare distribution.

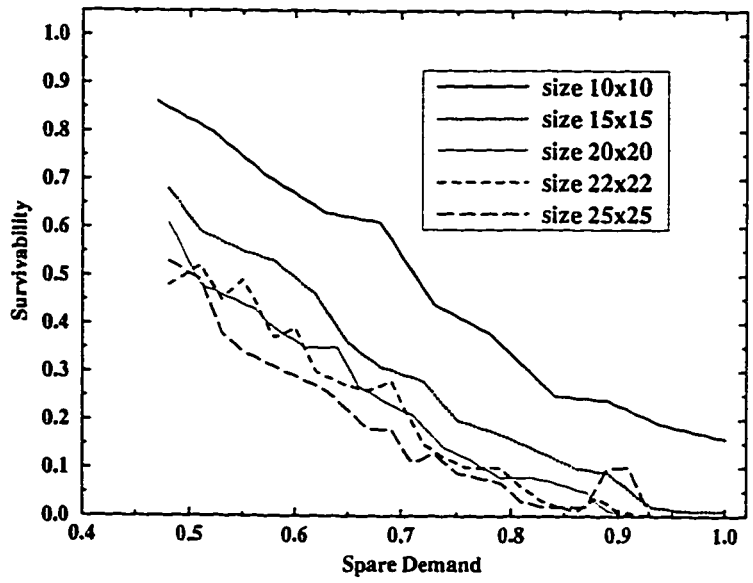


Figure 7.7: Variation in survivability of the array for increasing size of the array for greedy reconfiguration algorithm using pattern based spare distribution.

Spare demand	10x10	15x15	20x20	22x22	25x25
0.50	86	64	56	50	50
0.75	42	20	13	11	9
1.00	16	1	0	0	0

Table 7.7: Variation in survivability of the array for increasing size of the array for greedy reconfiguration algorithm using pattern based spare distribution.

increasing size of the array for all values of spare demand.

Results in tabular form for incremental placement approach are given in Table 7.8.

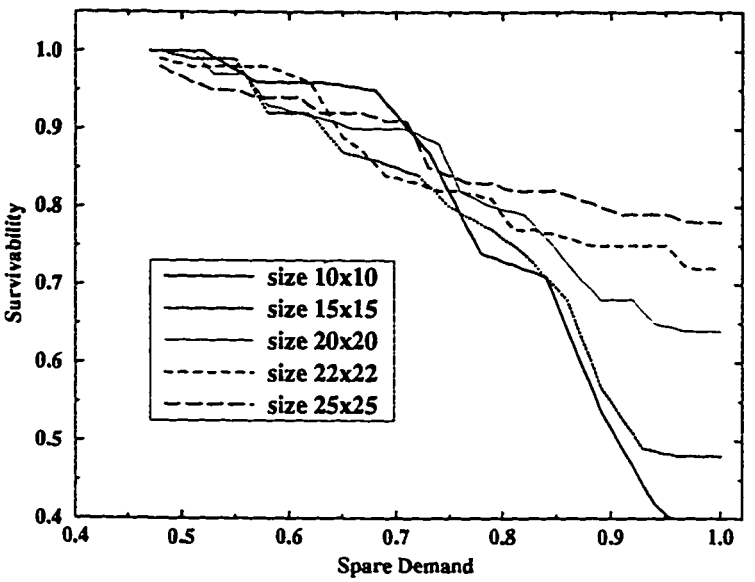


Figure 7.8: Variation in survivability of the array for increasing size of the array for incremental distance reconfiguration algorithm using pattern based spare distribution.

From Table 7.8 we see a different trend in variations in survivability of the array

Spare demand	10x10	15x15	20x20	22x22	25x25
0.50	100	100	100	99	98
0.75	81	80	85	82	84
1.00	34	48	64	72	78

Table 7.8: Variation in survivability of the array for increasing size of the array for incremental distance reconfiguration algorithm using pattern based spare distribution.

with increasing spare demand and size of the array. We see that as the size of the array increases survivability increases for higher values of spare demand. This is due to the nature of the sequence of moves taken. As the size of the array grows spare elements get further away from the impaired elements. Thus the reconfiguration strategy is more efficiently utilized.

7.5 Discussion

The framework for yield enhancement of processor arrays has been developed with the objective of providing an integrated tool to simulate the optimal processor array element redundancy and switch bus interconnection redundancy based on defect distribution obtained from defect models.

A range of reconfiguration algorithms can be applied in order to reconfigure faulty processor arrays. To make the framework versatile enough to adapt to different reconfiguration schemes, processing element redundancy and interconnection redundancy, it has been developed in three layers.

The first layer supports reconfiguration of different interconnection fabrics. Two important types of switched bus interconnections which have been used earlier are 3×3 and 3×2. Comparison of results obtained from simulations performed on the framework indicate that survivability of 3×2 interconnection resources is fairly high

for semi-clustered faults. Further improvement is expected if more advanced reconfiguration schemes are applied. Therefore, there is a good potential of reducing cost of manufacture of WSI fault tolerant processor arrays by reducing interconnection resources.

The second layer has utilities which support search for suitable spares for impaired (faulty) elements in processor arrays. These utilities are useful to restrict reconfiguration within user defined limits. They also allow the user control over the depth of search to find a suitable spare for impaired elements. The two utilities are the tree search and four path search. Results obtained from simulation indicate that in tree search survivability figures are better than for four path search. However, the time required to search is of exponential order. Therefore, for large size arrays time required for searching becomes a bottle neck.

To establish versatality of the framework two reconfiguration schemes are implemented in the third layer of the framework. These techniques are the greedy approach and incremental distance approach. Application of these techniques paves way for application of other deterministic global routing algorithms.

Comparison of results obtained from simulation for $\alpha = 1$ are shown in Table 7.9. Results indicate that incremental distance algorithm performs much better than greedy algorithm. This is due to the fact that incremental distance algorithm exploits search mechanism in a much better way than the greedy algorithm. It adopts a restraint policy as opposed to a greedy policy adopted by the greedy algorithm,

Spare demand	Knowledge-based initial solution						Pattern-based initial solution					
	Incremental distance algo.						10x10		15x15		20x20	
	10x10		15x15		20x20		greedy	incre-mental	greedy	incre-mental	greedy	incre-mental
	3x3	3x2	3x3	3x2	3x3	3x2	3x3	3x3	3x3	3x3	3x3	3x3
0.50	100	100	100	100	100	100	80	100	68	100	61	100
0.75	100	99.3	100	94	100	91	41	80	20	80	13	85
1.00	93	68	90	46	84	29	16	34	1	48	0	51

Table 7.9: A comparison of array survivability of proposed techniques.

thus preventing moves of impaired elements which block the way for reconfiguration of other impaired elements.

Using the three layered framework we can start reconfiguration from an initial solution which is pattern based or from an initial solution which is knowledge based. Pattern based spare distributions represent the class of reconfiguration schemes which assume an initial placement of logical elements [13]. The knowledge based spare distribution represents that class of reconfiguration algorithms which incrementally place the logical cells while eliminating the faulty elements [4]. Since, pattern based spare distribution techniques start from a known initial solution no computation overhead is required. On the contrary knowledge based spare distribution requires time complexity of cubic order to obtain the initial solution. Due to the added computation overhead, the initial solution has almost all the faulty elements eliminated yet the placement of logical cells is structured. Therefore, figures of survivability of knowledge based spare distribution are better than pattern based spare distribution, especially for higher values of spare demand.

To evaluate degradation in survivability due to heavy clustering of faults in proces-

sor arrays, we simulated survivability for semi-clustered and heavily clustered faults in processor arrays using knowledge based spare distribution. Comparison of results indicate that survivability falls sharply both with the size of the array as well as the increasing spare demand. This indicates that fault clustering on processor arrays has a negative impact on survivability.

To demonstrate the utility of the framework we performed simulations to find degradation in survivability for large sized processor arrays. Simulation results indicate that greedy algorithm performs poorly to reconfigure large sized arrays (25×25). It experiences a steady decrease in its performance as the size of the array increases. Incremental placement technique using knowledge based spare distribution experiences a slight degradation in survivability with increasing size of the array for higher values of spare demand. This is expected as the formation of initial solution preserves the structure of placement of logical elements. Therefore, the few impaired elements can be easily reconfigured.

Incremental distance algorithm using pattern based spare distribution shows a different trend in survivability with increasing size of the array. Survivability falls slightly with increasing area for lower values of spare demand. But for higher values of spare demand survivability improves with increasing size of the array. This trend may be due to the fact that in smaller sized arrays impaired elements are more closely spaced than in large sized arrays. Thus, causing reconfiguration of impaired elements within the first few passes thus blocking successful reconfiguration of the re-

maining impaired elements which have to be reconfigured using longer path threads.

7.6 Concluding Remarks

In this chapter simulation results related to different layers and modules of the framework were presented. Comparison of performance of alternative methods in some of the layers were shown to impress the reader with the relative advantages and disadvantages of each technique. As an example to demonstrate the utility of this framework simulations were conducted using different technique for large sized arrays.

Results of survivability of processor arrays with 3×2 interconnection resources show good performance indicating the potential of their use.

It is found that survivability of incremental distance algorithm increases with increasing size of the array. Thus showing its applicability for large size arrays.

Chapter 8

Conclusions and Future Work

Processor arrays are suitable for WSI/VLSI technology because of the regularity, local interconnection and pipeline capability. A number of computationally intensive problems can be executed using processor arrays. These include matrix operations, image processing and signal processing application etc. However, due to their large size yield at the time of manufacture is very low. Therefore, redundant elements and reconfiguration hardware have to be introduced on the chip to compensate for the occurence of faults on wafers/chips. On the other hand, it has been observed that yield falls sharply with increasing chip area. This indicates that the chip can be designed around an optimal amount of additional logic on the chip to improve the yield.

Existing reconfiguration techniques can be classified into four categories. They are local reconfiguration schemes, hierarchical schemes, hybrid schemes and global re-

configuration schemes. Local reconfiguration schemes have very high percentage of redundancy and have low survivability. Though hierarchical reconfiguration schemes have lower percentage of redundancy than local reconfiguration schemes, the percentage of overall redundancy is very high. Hybrid reconfiguration schemes have lesser percentage of redundancy than local and hierarchical reconfiguration schemes but survivability is lesser than global reconfiguration techniques.

Global reconfiguration techniques have low redundancy and high percentage of survivability. However, reconfiguration mechanism is complex. Most of the existing techniques are tied to some interconnection mechanism. Therefore, existing global reconfiguration techniques cannot be applied to the range of interconnection switch fabrics to determine the optimal redundancy requirement.

In this thesis we have developed a framework for yield enhancement of processor arrays which allows chip designers to predict survivability of the processor array with different combinations of spare element and interconnection resource redundancy. Given the size of the physical array and the size of the logical array we can obtain measure of survivability for different types of interconnection resources.

The rest of the chapter is organized as follows. In section 8.1 we present the conclusions of the thesis and in section 8.2 some future extensions of this work are presented.

8.1 Conclusions

The framework for yield enhancement of processor arrays is a three layer model with two extra modules to provide the neccessary support for simulations. The three layers are the rules platform, spare search techniques and policy based algorithms for sequence of moves. The two modules are spare distribution for initial solution formation module and the fault distribution module.

The rules platform is designed to provide reconfiguration through fault avoidance on 3×3, 3×2, 2×3 and 2×2 switch bus interconnection. Simulation were conducted to compare the performance of 3×3 and 3×2. For semi-clustered faults survivability of 3×2 interconnections is above 90%. The results indicate its potential use in yield enhancement of processor arrays.

The second layer of the framework provides different levels of search for suitable spares to be assigned to impaired (faulty) elements. Due to very high complexity of complete tree search (exponential order), we have implemented an alternative search technique called the four-path search. The four-path search checks moves only on four paths with maximum probability of a successful move. Comparison of their relative performance is shown.

Two reconfiguration techniques were implemented in the third layer. The two reconfiguration approaches are the greedy approach and the incremental distance approach. The incremental distance approach is of higher computation complexity but

has relatively better performance.

The module for formation of initial solution performs its task by either making use of the knowledge of fault locations on the processor array or not. If the knowledge of fault locations is used then technique with order of complexity of $O(N^3)$ have to be used to form the initial solution. If use of knowledge of fault locations is not used then standard pattern based initial solutions can be directly used for reconfiguration thus requiring no computation. Comparison of the performance of the two approaches shows that knowledge based formation of initial solution has better chances of survivability than pattern based approach.

The other module developed was the fault distribution module. It makes fault distribution on the array using the large-size defect distribution model. It uses the clustering parameter α and defects density λ to make defect distribution on the array. Simulations were conducted to determine survivability of processor arrays for semi-clustered ($\alpha = 1.0$) and heavily clustered faults ($\alpha = 0.3$). Comparison indicates that survivability falls with increasing clustering tendency.

We conclude this thesis with the comment that framework for yield estimation is a powerful tool that would aid chip/wafer designers of processor arrays to make better decisions to estimate the optimal amount of spare element and interconnection resource redundancy for obtaining better yield.

8.2 Future Work

In this thesis the structure of a framework for yield enhancement of processor arrays has been proposed. The framework is flexible and many extensions can be made to make a powerful tool for yield estimation of processor arrays.

Various types of rules can be incorporated in the core to account for the different figures of merrits. One important figure of merrit is the maximum interconnection delay in the processor array. Simple rules can limit the maximum interconnection length at the time of reconfiguration thus ensuring that the final solution has the desired maximum interconnection lenght.

In order to have a bigger range of switch bus interconnections rules for 4 tracks should be formulated and implemented.

An important factor which determines the efficiency of various reconfiguration mechanisms is the sequence of moves. It is not known whether all the instances of the FFA are obtainable from each other through a sequence of moves. If such a sequence can be found then more efficient reconfiguration schemes can be formulated.

Though survivability is a good measure to predict yield it is not the exact measure. To obtain accurate figures of yield better defect models should be used and faults in switches should be taken into consideration. More efficient defect models should have exact spatial distribution of faults. This is necessary to determine the failures of switches. To tolerate faults in switches rules should be made to reconfigure

through defective switch avoidance.

Appendix A

Assignment rules for 2×1 column resources and 1×2 row resources are given below. Only the row assignment rules are given, column assignment can be obtained similarly.

Define *inverse row range* of a physical cell, denoted by ϕ^r, as the limited range of a logical row which can be assigned to the physical cell. When a logical cell is assigned to a physical cell, the assignment affects other assignments to the neighboring cells due to limited interconnection resources. Let's denote the assignment of a logical cell L to a physical cell P as $L-to-P$. The effect of $L-to-P$ on a neighboring physical cells, say G is represented by $\phi^r[L-to-P, G] = r_{min}/r_{max}$. This indicates that only the logical row α for $r_{min} \leq \alpha \leq r_{max}$ can be assigned to G due to the assignment $L-to-P$. That is $r_{min}(r_{max})$ is the minimum (maximum) logical row index which can be assigned to G due to the assignment $L-to-P$. In other words, if G is already assigned to a logical row which is not in the *inverse-row-range*, the assignment $L-to-P$ will cause resource conflict, thus the logical cell L cannot be assigned to P. r_{min} is denoted by $\phi^r_{min}[L-to-P, G]$, and r_{max} is denoted by $\phi^r_{max}[L-to-P, G]$. If a cell G is faulty $\phi^r[L-to-P, G] = r_{min}/r_{max}$ means that the logical row α for $\alpha \leq r_{min}$ cannot be connected above G, and the logical row α for $\alpha \geq r_{max}$ cannot be connected below G.

Forward Assignment Rule

Suppose that $L = [i,j]_L$ assigned to P makes a forward row assignment. See figure .1(a) . $\phi^r[L - to - P, Q]$ is determined as follows. First, consider the case that Q is active. If a logical row $i - 3$ is connected to Q, then conflict between logical rows $i - 1$ and $i - 2$ will occurs. This is because two logical rows cannot be connected through only one horizontal track. Therefore, logical row $i - 2$ is the minimum logical row which can be assigned to Q. Since logical row i is assigned to P, logical row $i - 1$ is the maximum logical row index which can be assigned to Q. Therefore, if Q is active, $\phi^r[L - to - P, Q] = i - 2/i - 1$. See figure .2(a)

Next consider the case that Q is passive. If logical row $i - 2$ is connected below Q, then conflict between logical rows $i - 1$ and $i - 2$ will occur due to the existence of only one horizontal track between Q and P. Therefore, a logical row whose index is equal to or smaller than $i - 2$ must not be connected below Q. Since logical row i is connected to P, a logical row whose index is equal to or larger than i must not be connected above Q. Therefore, when Q is passive, $\phi^r[L - to - P, Q] = i - 2/i$. Similarly, $\phi^r[L - to - P, B]$ can be determined.

Let's determine $\phi^r[L - to - P, R]$. If logical row $i - 2$ is connected below R, conflict among logical rows i, and will occur in H-gate (P,R), since only two vertical tracks are available for three row connections. If logical row $i + 2$ is connected above R, conflict among logical rows i, $i+1$, and $i+2$ will also occur in H-gate (P,R). Thus, a

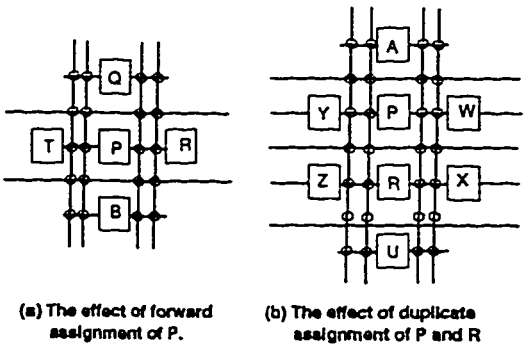


Figure .1: Forward assignment and duplicate assignment.

logical row whose index is equal to or greater than $i+2$ must not be connected above R. Therefore, $\phi^r[L-to-P,R] = i-2/i+2$. Similarly, $\phi^r[L-to-P,T] = i-2/i+2$. See figure .2(b).These observations lead to the following forward assignment rule.

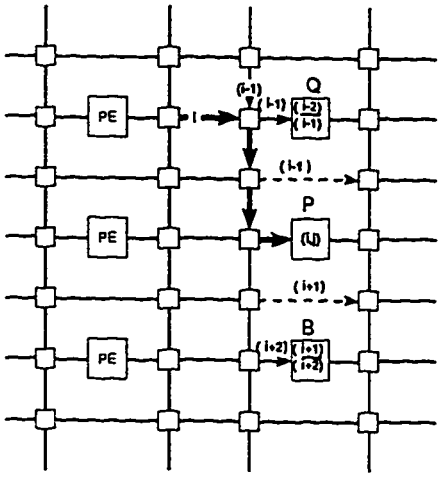
Formal description of the forward assignment rule.

Suppose, $L = [i,j]_L$ assigned to P makes a forward row assignments. The ϕ^r's of four neighboring cells Q, B, R, and T are determined by the assignment of $L-to-P$ as follows:

1. if Q is active, $\phi^r[L-to-P,Q] = i-2/i-1$, else $\phi^r[L-to-P,Q] = i-2/i$.

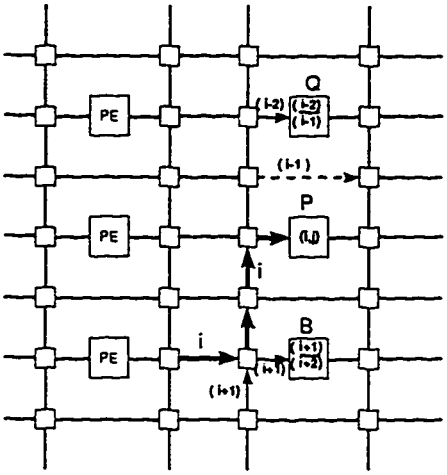
2. if B is active, $\phi^r[L-to-P,B] = i = 1/i+2$, else $\phi^r[L-to-P,B] = i/i+2$.

3. $\phi^r[L-to-P,R] = \phi^r[L-to-P,T] = i-2/i+2$.



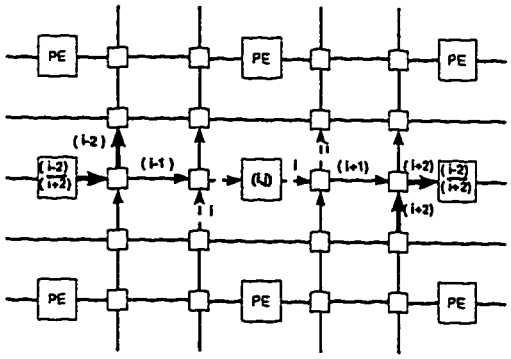
(i) If Q is active $\phi^r[L_p,Q] = (i\text{-}2/i\text{-}1)$, else $\phi^r[L_p,Q] = (i\text{-}2/i)$.

(ii) If B is active $\phi^r[L_p,B] = (i+2/i+1)$, else $\phi^r[L_p,B] = (i+2/i)$.

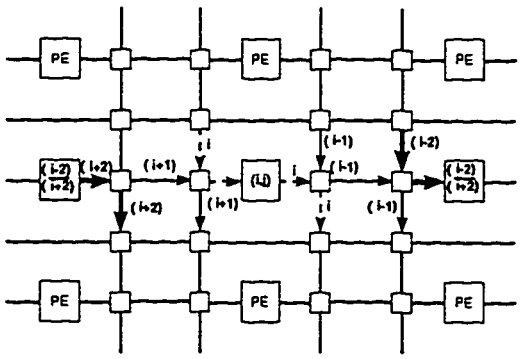


(i) If Q is active $\phi^r[L_p,Q] = (i\text{-}2/i\text{-}1)$, else $\phi^r[L_p,Q] = (i\text{-}2/i)$.

(ii) If B is active $\phi^r[L_p,B] = (i+2/i+1)$, else $\phi^r[L_p,B] = (i+2/i)$.



(iii) $\phi^r[L_p,R] = \phi^r[L_p,T] = (i\text{-}2/i+1)$.



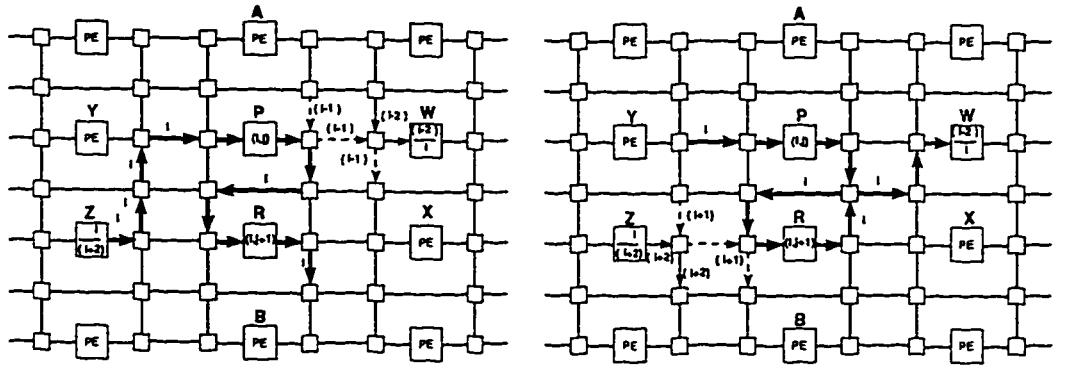
(iii) $\phi^r[L_p,R] = \phi^r[L_p,T] = (i\text{-}2/i+1)$.

Figure .2: Explanation of the forward assignment rule.

Duplicated Assignment Rule

Suppose that $L_P = [i,j]_L$ assigned to P constitutes a duplicated row assignment with $L_R = [i,j+1]_L$ assigned to R (see figure .1(b)). $\phi^r[L-to-P,W]$ is determined as follows. First, consider the case that W is active. If logical row $i-3$ is assigned to$W$, then conflict among logical rows $i, i-1$, and $i-2$ will occur in H-gate(P,W). Hence the minimum logical row which can be assigned to W is $i-2$. If logical row $i+1$ is assigned to W, then conflict between logical rows i and $i+1$ occurs in H-gate(P,W), since logical rows i and $i+1$ require 2 and 1 vertical tracks, respectively, in H-gate(P,W). Thus, the maximum row index which can be assigned to W is i. Therefore, if W is active, $\phi^r[L-to-P,W] = i-2/i$. See figure .3.

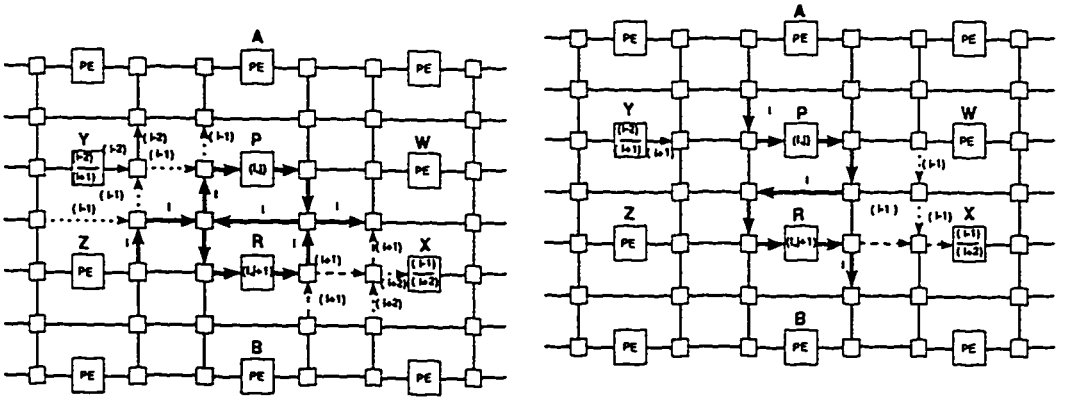
Next, consider the case that W is passive. If logical row $i-2$ is connected below W, conflict will occur in H-gate(P,W), hence $\phi^r[L-to-P,W] = i-2/i$. If logical row $i+1$ is connected above W, a conflict will occur in H-gate(P,W), hence $\phi^r_{max}[L-to-P,W] = i+1$. Therefore, if W is passive, $\phi^r[L-to-P,W] = i-2/i+1$. In a similar way, ϕ^r's of other cells can be determined as shown in figure .3 and figure .4. The rules for the top and the bottom cells A and B is the same as the rules for the top and bottom cells in forward assignment. Thus, the duplicated assignment rule is as given below.



(a) The low row ranges of right-top and left-bottom cells W and Z of the duplicate assigned cells P and Q. W and Z are non-faulty.

(b) The high row ranges of right-top and left-bottom cells W and Z of the duplicate assigned cells P and Q. W and Z are non-faulty.

Figure .3: Effect of duplicate assignment of the cells P and R on the right-top and left-bottom cells .



(c) The row ranges of left-top and right-bottom cells Y and X of the duplicate assigned cells P and Q. X and Y are non-faulty.

(d) The high ranges of left-top and right-bottom cells Y and X of the duplicate assigned cells P and Q. X and Y are non-faulty.

Figure .4: Effect of duplicate assignment of the cells P and R on the right-bottom and left-top cells .

Formal description of the duplicated assignment rule

Suppose that $L = [i,j]_L$ assigned to P constitutes a duplicated row assignment with $L_R = [i,j+1]_L$ assigned to R. The ϕ^r's of the six neighboring cells are limited as follows.

1. If A is active, $\phi^r[L-to-P,A] = i-2/i = 1$, else $\phi^r[L-to-P,A] = i-2/i$.

2. If U is active, $\phi^r[L-to-P,U] = i+1/i+2$, else $\phi^r[L-to-P,U] = i/i+2$.

3. If W is active, $\phi^r[L-to-P,W] = i = 2/i$, else $\phi^r[L-to-P,W] = i-2/i+1$

4. If X is active, $\phi^r[L-to-P,X] = i-1/i+2$, else $\phi^r[L-to-P,X] = i-2/i+2$

5. If Y is active, $\phi^r[L-to-P,Y] = i-2/i+1$, else $\phi^r[L-to-P,Y] = i-2/i+2$.

6. If Z is active, $\phi^r[L-to-P,Z] = i/i+2$, else $\phi^r[L-to-P,Z] = i-1/i+2$.

Propagation Rule

Suppose that $L = [i,j]_L$ assigned to P constitutes a forward row assignment (see figure .5). Neighboring cells of P whose ϕ^r's are affected due to the assignment$L-to-P$, are called *affected cells* of P, and P is called the *affecting cell.* If four neighboring cells of P (i.e., Q,R,B, and T in figure .5) are all active, only these four cells belong to the affected cells. If any one of the four neighboring cells is not active,

the assignment effect is propagated vertically upward or downward until an active cell is encountered. In figure .5 upward propagations are required from T to C and from Q to A, and a downward propagation is from T to D. An upward propagation can be explained similarly.

Propagation Rule

Case 1: ($P_i = [i,n]_P$ and $P_{i-1} = [i-1,n]_P$ are in upward propagation). If P_{i-1} is passive, $\phi^r[L - to - P, P_{i-1}] = r_{min} - 1/r_{max}$, else $\phi^r[L - to - P, P_{i-1}] = r_{min} - 1/r_{max} - 1$.

Case 2: ($P_i = [i,n]_P$ and $P_{i+1} = [i+1,n]_P$ are in downward propagation). If P_{i-1} is passive, $\phi^r[L - to - P, P_{i+1}] = r_{min}/r_{max} + 1$, else $\phi^r[L - to - P, P_{i+1}] = r_{min} + 1/r_{max} + 1$.

Skewed Rule

If a logical row is connected through clustered passive cells, a logical row must be allocated to a proper V-gate to ensure conflict-free connections.

Definition:

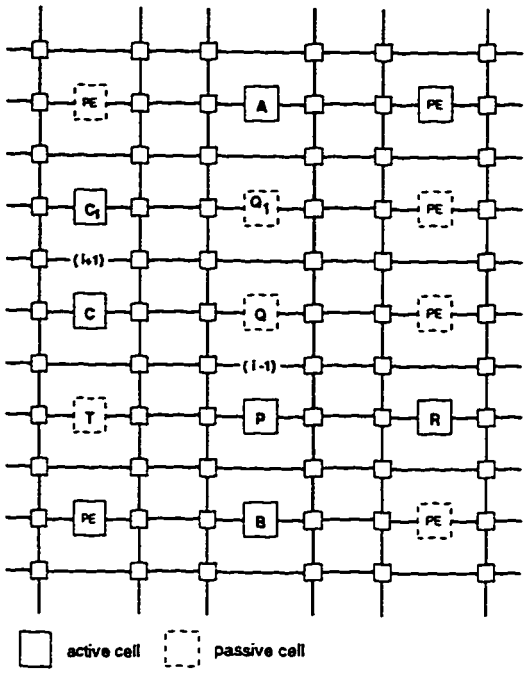


Figure .5: Affected cells of a forward assignment of P.

Consider logical rows i and $i+k$. Suppose that logical row i is allocated to a V-gate V_{g1} in a physical column m. Suppose that logical row $i+k$ is connected through a V-gate V_{g2} in the physical column $m-1$ or $m+1$. If the allocation of V_{g2} is lower than that of V_{g1}, it is said that the allocation of logical row i and logical row $i+k$ makes *skewed-level k*.

For example, in figure .5 suppose that logical rows $i+1$ and $i-1$ are connected to V-gate (C_1, C) and V-gate (Q, P), respectively. The allocation of logical row $i+1$ to V-gate (C_1, C) and that of logical row $i-1$ to the V-gate (Q, P) makes skewed-level 2 causing resource conflict, since only two vertical tracks are available in a H-gate (C, Q) for three logical rows $i+1, i$ and $i-1$. To avoid the conflict from the skewed

condition, logical row $i-1$ must be allocated to a V-gate which is located above Q.

The Skewed Rule: Suppose that 1×2 row resources are given. Any two logical rows which are connected through a V-gate of clustered passive cells must not make skewed-level k for $k \geq 2$.

Array Reconfigurability

A cell in the array has one or more affecting cells, and thus the combined effect of affecting cells must be considered. For example, in figure .5, Q_1 is affected by cells P, A, C and C_1. Suppose that $L_p = [i, -]_L$, $L_A = [i-2, -]_L$, $L_C = [i-2, -]_L$ and $L_{C_1} = [i-3, -]_L$, where L_p, L_A, L_C and L_{C_1} are the logical cells of P, A, C and C_1, respectively. The - indicates any column. The ϕ^r's of passive cells in V-bar (A, P) are calculated by applying the forward assignment rules and propagation rules, and shown in Table. Let's derive the combined ϕ^r's of Q_1. The combined ϕ^r of Q_1 must satisfy the effects of all affecting cells. Thus the combined ϕ^r_{min} of Q_1 is $i-2$, which is the maximum value among ϕ^r_{min}'s of all affecting cells (see Table 8.2. In general, the combined effect of all affecting cells, say $P_1, P_2, \cdots, P_k$, on a cell G is denoted by $\phi^r[L_{com} - to - P_{com}, G]$, where $L_{com} = \{L_1, L_2, \cdots, L_k\}$ and $P_{com} = \{P_1, P_2, \cdots, P_k\}$. In the $\phi^r[L_{com} - to - P_{com}, G]$, the maximum ϕ^r_{min}(minimum ϕ^r_{max}) is defined as *effective* ϕ^r_{min}(*effective* ϕ^r_{max}), meaning that *effective* ϕ^r is used to decide the allowable range of logical rows for G. The *effective* ϕ^r_{min}'s for figure .5 are shown in table 1

physical cells α	ϕ^r[L-to-P, α]	ϕ^r[L-to-A, α]	ϕ^r[L-to-C, α]	ϕ^r[L-to-C_1, α]	effective ϕ^r
Q_1	(i-3/i)	(i-2/i)	(i-5/i)	(i-5/i-1)	(i-2/i-1)
Q	(i-2/i)	(i-2/i+1)	(i-4/i)	(i-5/i)	(i-2/i)

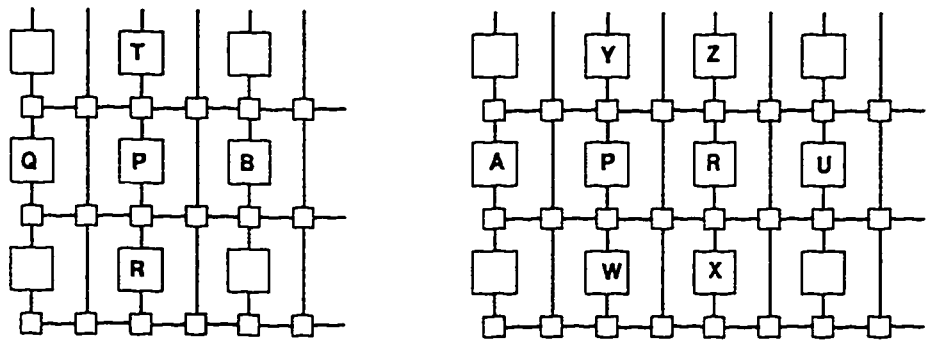
Table .1: ϕ^r's and effective ϕ^r's of passive cells in V-BAR (A,P) in Figure .5.

Theorem 1 : If all logical cells are assigned satisfying the assignment rules based on the 3 × 3 interconnection resources, the logical array can be connected through the 3 × 3 interconnection resources without resource conflicts, hence, the faulty array is successfully reconfigured.

The proof of Theorem 1 is given in [21].

The rules described so far are the rules applicable to the row interconnection of the processor array. The rules for the column interconnection can be obtained easily. The rules for reconfiguration changes if the 3 × 2 interconnection resources are used. The rules for the row interconnection remain the same as there is no change in the resources and the column interconnection resources and row interconnection resources are not shared. Description and definition of rules for 1 × 1 horizontal or vertical interconnection resources is not given in the work by [4]. But, these can be easily deduced from the the 1 × 2 interconnection resources. Formal description of the column interconnection rules is given below.

Forward assignment rule



(a) The effect of forward assignment of P.

(b) The effect of duplicate assignment of P and R.

Figure .6: Affected cells of a forward and duplicate assignment in 3x2 interconnection resources.

Suppose, $L = [i,j]_L$ assigned to P makes a forward column assignments. The ϕ^c's of four neighboring cells Q, B, R, and T (see figure .6(a)) are determined by the assignment of $L - to - P$ as follows:

1. if Q is active, $\phi^c[L-to-P,Q] = j-2/j-1$, else $\phi^c[L-to-P,Q] = j-2/j$.

2. if B is active, $\phi^c[L-to-P,B] = j+1/j+2$, else $\phi^c[L-to-P,B] = j/j+2$.

3. $\phi^c[L-to-P,R] = \phi^c[L-to-P,T] = j-1/j+1$.

Duplicated Assignment Rule

Suppose that $L = [i,j]_L$ assigned to P constitutes a duplicated row assignment with $L_R = [i+1,j]_L$ assigned to R. The ϕ^c's of the six neighboring cells (see figure .6(b))are limited as follows.

1. If A is active, $\phi^c[L-to-P,A] = j-2/j-1$, else $\phi^c[L-to-P,A] = j-2/j$.

2. If U is active, $\phi^c[L-to-P,U] = j+1/j+2$, else $\phi^c[L-to-P,U] = j/j+2$.

3. If W is active, $\phi^c[L-to-P,W] = j-1/j-1$, else $\phi^c[L-to-P,W] = j-1/j$

4. If X is active, $\phi^c[L-to-P,X] = j/j+1$, else $\phi^c[L-to-P,X] = j-1/j+1$

5. If Y is active, $\phi^c[L-to-P,Y] = j-1/j$, else $\phi^c[L-to-P,Y] = j-1/j+1$.

6. If Z is active, $\phi^c[L-to-P,Z] = j+1/j+1$, else $\phi^c[L-to-P,Z] = j/j+1$.

Propagation Rule

The propagation rule does not change as there is one column track between any two horizontally adjacent processing elements.

Skewed Rule

If a logical column is connected through clustered passive cells, a logical column must be allocated to a proper H-gate to ensure conflict-free connections.

The Skewed Rule: Suppose that 1×1 column resources are given. Any two logical columns which are connected through a H-gate of clustered passive cells must not make skewed-level k for $k \geq 1$.

Appendix B

In this appendix we illustrate the reconfiguration process of a large size processor array of size 15×15. Most of the steps that are taken are with distance greater than 2.

The initial solution is a plus pattern based initial solution. Column 8 and row 7 in Figure .7 are spare row and spare column, respectively. In Figure .8 all moves are made with manhattan distance 1. Four impaired elements are assigned to spare elements. $(1,9)_P$ is assigned to physical location $(1,8)_P$. $(2,9)_P$ is assigned to physical location $(2,8)_P$. $(9,3)_P$ is assigned to physical location $(8,3)_P$. $(13,9)_P$ is assigned to physical location $(13,8)_P$. Figure .9 shows moves taken with distance 2. Four impaired elements are assigned to physical elements. $(3,10)_P$ is assigned to physical location $(3,8)_P$. $(4,10)_P$ is assigned to physical location $(4,8)_P$. $(8,7)_P$ is assigned to physical location $(7,8)_P$ through $(8,8)_P$. $(9,8)_P$ is assigned to physical location $(8,7)_P$. In Figure .10 steps with distance 3 are shown. Two impaired elements are reconfigured. $(7,2)_P$ is assigned to $(8,4)_P$ through $(7,3)_P$ and $(7,4)_P$. $(11,9)_P$ is assigned to $(8,9)_P$. In Figure .11 four impaired elements are assigned to spare elements at distance 4. $(9,2)_P$ is assigned to $(8,5)_P$ through $(8,3)_P$ and $(8,4)_P$. $(12,10)_P$ is assigned to $(8,10)_P$. $(12,11)_P$ is assigned to $(8,11)_P$. $(12,9)_P$ is assigned to $(15,8)_P$. In Figure .12 one impaired element is assigned to a spare element at distance 5. $(2,10)_P$ is assigned to $(5,8)_P$. In Figure .13 one impaired element is assigned to a spare element at distance 6. $(3,11)_P$ is assigned to $(8,12)_P$.

In Figure .14 one impaired element is assigned to a spare element at distance 8. $(2,11)_P$ is assigned to $(8,13)_P$. In Figure .15 one impaired element is assigned to a spare element at distance 9. $(10,7)_P$ is assigned to $(8,14)_P$. The final configuration obtained after the last impaired element is assigned to a spare element at manhattan distance 11 is shown in Figure .15. $(1,11)_P$ is assigned to $(8,15)_P$.

1, 1	1, 2	1, 3	1, 4	1, 5	1, 6	1, 7	0, 0	1, 8	1, 9	1,10	1,11	1,12	1,13	1,14
2, 1	2, 2	2, 3	2, 4	2, 5	2, 6	2, 7	0, 0	2, 8	2, 9	2,10	2,11	2,12	2,13	2,14
3, 1	3, 2	3, 3	3, 4	3, 5	3, 6	3, 7	0, 0	3, 8	3, 9	3,10	3,11	3,12	3,13	3,14
4, 1	4, 2	4, 3	4, 4	4, 5	4, 6	4, 7	0, 0	4, 8	4, 9	4,10	4,11	4,12	4,13	4,14
5, 1	5, 2	5, 3	5, 4	5, 5	5, 6	5, 7	0, 0	5, 8	5, 9	5,10	5,11	5,12	5,13	5,14
6, 1	6, 2	6, 3	6, 4	6, 5	6, 6	6, 7	0, 0	6, 8	6, 9	6,10	6,11	6,12	6,13	6,14
7, 1	7, 2	7, 3	7, 4	7, 5	7, 6	7, 7	0, 0	7, 8	7, 9	7,10	7,11	7,12	7,13	7,14
0, 0	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0
8, 1	8, 2	8, 3	8, 4	8, 5	8, 6	8, 7	0, 0	8, 8	8, 9	8,10	8,11	8,12	8,13	8,14
9, 1	9, 2	9, 3	9, 4	9, 5	9, 6	9, 7	0, 0	9, 8	9, 9	9,10	9,11	9,12	9,13	9,14
10, 1	10, 2	10, 3	10, 4	10, 5	10, 6	10, 7	0, 0	10, 8	10, 9	10,10	10,11	10,12	10,13	10,14
11, 1	11, 2	11, 3	11, 4	11, 5	11, 6	11, 7	0, 0	11, 8	11, 9	11,10	11,11	11,12	11,13	11,14
12, 1	12, 2	12, 3	12, 4	12, 5	12, 6	12, 7	0, 0	12, 8	12, 9	12,10	12,11	12,12	12,13	12,14
13, 1	13, 2	13, 3	13, 4	13, 5	13, 6	13, 7	0, 0	13, 8	13, 9	13,10	13,11	13,12	13,13	13,14
14, 1	14, 2	14, 3	14, 4	14, 5	14, 6	14, 7	0, 0	14, 8	14, 9	14,10	14,11	14,12	14,13	14,14

Figure .7: Plus pattern initial solution. Dashed boxes indicate faulty elements.

1, 1	1, 2	1, 3	1, 4	1, 5	1, 6	1, 7	1, 8	0, 0	1, 9	1,10	1,11	1,12	1,13	1,14
2, 1	2, 2	2, 3	2, 4	2, 5	2, 6	2, 7	2, 8	0, 0	2, 9	2,10	2,11	2,12	2,13	2,14
3, 1	3, 2	3, 3	3, 4	3, 5	3, 6	3, 7	0, 0	3, 8	3, 9	3,10	3,11	3,12	3,13	3,14
4, 1	4, 2	4, 3	4, 4	4, 5	4, 6	4, 7	0, 0	4, 8	4, 9	4,10	4,11	4,12	4,13	4,14
5, 1	5, 2	5, 3	5, 4	5, 5	5, 6	5, 7	0, 0	5, 8	5, 9	5,10	5,11	5,12	5,13	5,14
6, 1	6, 2	6, 3	6, 4	6, 5	6, 6	6, 7	0, 0	6, 8	6, 9	6,10	6,11	6,12	6,13	6,14
7, 1	7, 2	7, 3	7, 4	7, 5	7, 6	7, 7	0, 0	7, 8	7, 9	7,10	7,11	7,12	7,13	7,14
0, 0	0, 0	8, 3	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0
8, 1	8, 2	0, 0	8, 4	8, 5	8, 6	8, 7	0, 0	8, 8	8, 9	8,10	8,11	8,12	8,13	8,14
9, 1	9, 2	9, 3	9, 4	9, 5	9, 6	9, 7	0, 0	9, 8	9, 9	9,10	9,11	9,12	9,13	9,14
10, 1	10, 2	10, 3	10, 4	10, 5	10, 6	10, 7	0, 0	10, 8	10, 9	10,10	10,11	10,12	10,13	10,14
11, 1	11, 2	11, 3	11, 4	11, 5	11, 6	11, 7	0, 0	11, 8	11, 9	11,10	11,11	11,12	11,13	11,14
12, 1	12, 2	12, 3	12, 4	12, 5	12, 6	12, 7	12, 8	0, 0	12, 9	12,10	12,11	12,12	12,13	12,14
13, 1	13, 2	13, 3	13, 4	13, 5	13, 6	13, 7	0, 0	13, 8	13, 9	13,10	13,11	13,12	13,13	13,14
14, 1	14, 2	14, 3	14, 4	14, 5	14, 6	14, 7	0, 0	14, 8	14, 9	14,10	14,11	14,12	14,13	14,14

Figure .8: Moves with distance 1.

1,1	1,2	1,3	1,4	1,5	1,6	1,7	1,8	0,0	1,9	1,10	1,11	1,12	1,13	1,14
2,1	2,2	2,3	2,4	2,5	2,6	2,7	2,8	0,0	2,9	2,10	2,11	2,12	2,13	2,14
3,1	3,2	3,3	3,4	3,5	3,6	3,7	3,8	3,9	0,0	3,10	3,11	3,12	3,13	3,14
4,1	4,2	4,3	4,4	4,5	4,6	4,7	4,8	4,9	0,0	4,10	4,11	4,12	4,13	4,14
5,1	5,2	5,3	5,4	5,5	5,6	5,7	0,0	5,8	5,9	5,10	5,11	5,12	5,13	5,14
6,1	6,2	6,3	6,4	6,5	6,6	6,7	7,7	6,8	6,9	6,10	6,11	6,12	6,13	6,14
7,1	7,2	7,3	7,4	7,5	7,6	0,0	0,0	7,8	7,9	7,10	7,11	7,12	7,13	7,14
0,0	0,0	8,3	0,0	0,0	8,6	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
8,1	8,2	0,0	8,4	8,5	8,7	0,0	0,0	8,8	8,9	8,10	8,11	8,12	8,13	8,14
9,1	9,2	9,3	9,4	9,5	9,6	9,7	0,0	9,8	9,9	9,10	9,11	9,12	9,13	9,14
10,1	10,2	10,3	10,4	10,5	10,6	10,7	0,0	10,8	10,9	10,10	10,11	10,12	10,13	10,14
11,1	11,2	11,3	11,4	11,5	11,6	11,7	0,0	11,8	11,9	11,10	11,11	11,12	11,13	11,14
12,1	12,2	12,3	12,4	12,5	12,6	12,7	12,8	0,0	12,9	12,10	12,11	12,12	12,13	12,14
13,1	13,2	13,3	13,4	13,5	13,6	13,7	0,0	13,8	13,9	13,10	13,11	13,12	13,13	13,14
14,1	14,2	14,3	14,4	14,5	14,6	14,7	0,0	14,8	14,9	14,10	14,11	14,12	14,13	14,14

Figure .9: Moves with distance 2.

1,1	1,2	1,3	1,4	1,5	1,6	1,7	1,8	0,0	1,9	1,10	1,11	1,12	1,13	1,14
2,1	2,2	2,3	2,4	2,5	2,6	2,7	2,8	0,0	2,9	2,10	2,11	2,12	2,13	2,14
3,1	3,2	3,3	3,4	3,5	3,6	3,7	3,8	3,9	0,0	3,10	3,11	3,12	3,13	3,14
4,1	4,2	4,3	4,4	4,5	4,6	4,7	4,8	4,9	0,0	4,10	4,11	4,12	4,13	4,14
5,1	5,2	5,3	5,4	5,5	5,6	5,7	0,0	5,8	5,9	5,10	5,11	5,12	5,13	5,14
6,1	6,2	6,3	6,4	6,5	6,6	6,7	7,7	6,8	6,9	6,10	6,11	6,12	6,13	6,14
7,1	0,0	7,2	7,3	7,5	7,6	0,0	0,0	7,8	7,9	7,10	7,11	7,12	7,13	7,14
0,0	0,0	8,3	7,4	0,0	8,6	0,0	0,0	8,8	0,0	0,0	0,0	0,0	0,0	0,0
8,1	8,2	0,0	8,4	8,5	8,7	0,0	0,0	9,8	8,9	8,10	8,11	8,12	8,13	8,14
9,1	9,2	9,3	9,4	9,5	9,6	9,7	0,0	10,8	9,9	9,10	9,11	9,12	9,13	9,14
10,1	10,2	10,3	10,4	10,5	10,6	10,7	0,0	0,0	10,9	10,10	10,11	10,12	10,13	10,14
11,1	11,2	11,3	11,4	11,5	11,6	11,7	0,0	11,8	11,9	11,10	11,11	11,12	11,13	11,14
12,1	12,2	12,3	12,4	12,5	12,6	12,7	12,8	0,0	12,9	12,10	12,11	12,12	12,13	12,14
13,1	13,2	13,3	13,4	13,5	13,6	13,7	0,0	13,8	13,9	13,10	13,11	13,12	13,13	13,14
14,1	14,2	14,3	14,4	14,5	14,6	14,7	0,0	14,8	14,9	14,10	14,11	14,12	14,13	14,14

Figure .10: Moves with distance 3.

1,1	1,2	1,3	1,4	1,5	1,6	1,7	1,8	0,0	1,9	1,10	1,11	1,12	1,13	1,14
2,1	2,2	2,3	2,4	2,5	2,6	2,7	2,8	0,0	2,9	2,10	2,11	2,12	2,13	2,14
3,1	3,2	3,3	3,4	3,5	3,6	3,7	3,8	3,9	0,0	3,10	3,11	3,12	3,13	3,14
4,1	4,2	4,3	4,4	4,5	4,6	4,7	4,8	4,9	0,0	4,10	4,11	4,12	4,13	4,14
5,1	5,2	5,3	5,4	5,5	5,6	5,7	0,0	5,8	5,9	5,10	5,11	5,12	5,13	5,14
6,1	6,2	6,3	6,4	6,5	6,6	6,7	7,7	6,8	6,9	6,10	6,11	6,12	6,13	6,14
7,1	0,0	7,2	7,3	7,5	7,6	0,0	0,0	7,8	7,9	7,10	7,11	7,12	7,13	7,14
0,0	0,0	8,2	8,3	7,4	8,6	0,0	0,0	8,8	8,9	8,10	0,0	0,0	0,0	0,0
8,1	0,0	0,0	8,4	8,5	8,7	0,0	0,0	9,8	9,9	9,10	8,11	8,12	8,13	8,14
9,1	9,2	9,3	9,4	9,5	9,6	9,7	0,0	10,8	10,9	10,10	9,11	9,12	9,13	9,14
10,1	10,2	10,3	10,4	10,5	10,6	10,7	0,0	0,0	11,9	11,10	10,11	10,12	10,13	10,14
11,1	11,2	11,3	11,4	11,5	11,6	11,7	0,0	0,0	0,0	0,0	11,11	11,12	11,13	11,14
12,1	12,2	12,3	12,4	12,5	12,6	12,7	12,8	0,0	12,9	12,10	12,11	12,12	12,13	12,14
13,1	13,2	13,3	13,4	13,5	13,6	13,7	0,0	11,8	13,9	13,10	13,11	13,12	13,13	13,14
14,1	14,2	14,3	14,4	14,5	14,6	14,7	14,8	13,8	14,9	14,10	14,11	14,12	14,13	14,14

Figure .11: Moves with distance 4.

1,1	1,2	1,3	1,4	1,5	1,6	1,7	1,8	0,0	1,9	1,10	1,11	1,12	1,13	1,14
2,1	2,2	2,3	2,4	2,5	2,6	2,7	2,8	0,0	0,0	2,10	2,11	2,12	2,13	2,14
3,1	3,2	3,3	3,4	3,5	3,6	3,7	3,8	2,9	0,0	3,10	3,11	3,12	3,13	3,14
4,1	4,2	4,3	4,4	4,5	4,6	4,7	4,8	3,9	0,0	4,10	4,11	4,12	4,13	4,14
5,1	5,2	5,3	5,4	5,5	5,6	5,7	5,8	4,9	5,9	5,10	5,11	5,12	5,13	5,14
6,1	6,2	6,3	6,4	6,5	6,6	6,7	7,7	6,8	6,9	6,10	6,11	6,12	6,13	6,14
7,1	0,0	7,2	7,3	7,5	7,6	0,0	0,0	7,8	7,9	7,10	7,11	7,12	7,13	7,14
0,0	0,0	8,2	8,3	7,4	8,6	0,0	0,0	8,8	8,9	8,10	0,0	0,0	0,0	0,0
8,1	0,0	0,0	8,4	8,5	8,7	0,0	0,0	9,8	9,9	9,10	8,11	8,12	8,13	8,14
9,1	9,2	9,3	9,4	9,5	9,6	9,7	0,0	10,8	10,9	10,10	9,11	9,12	9,13	9,14
10,1	10,2	10,3	10,4	10,5	10,6	10,7	0,0	0,0	11,9	11,10	10,11	10,12	10,13	10,14
11,1	11,2	11,3	11,4	11,5	11,6	11,7	0,0	0,0	0,0	0,0	11,11	11,12	11,13	11,14
12,1	12,2	12,3	12,4	12,5	12,6	12,7	12,8	0,0	12,9	12,10	12,11	12,12	12,13	12,14
13,1	13,2	13,3	13,4	13,5	13,6	13,7	0,0	11,8	13,9	13,10	13,11	13,12	13,13	13,14
14,1	14,2	14,3	14,4	14,5	14,6	14,7	14,8	13,8	14,9	14,10	14,11	14,12	14,13	14,14

Figure .12: Moves with distance 5.

1, 1	1, 2	1, 3	1, 4	1, 5	1, 6	1, 7	1, 8	0, 0	1, 9	1,10	1,11	1,12	1,13	1,14
2, 1	2, 2	2, 3	2, 4	2, 5	2, 6	2, 7	2, 8	0, 0	0, 0	2,10	2,11	2,12	2,13	2,14
3, 1	3, 2	3, 3	3, 4	3, 5	3, 6	3, 7	3, 8	2, 9	0, 0	0, 0	3,10	3,12	3,13	3,14
4, 1	4, 2	4, 3	4, 4	4, 5	4, 6	4, 7	4, 8	3, 9	0, 0	4,10	3,11	4,12	4,13	4,14
5, 1	5, 2	5, 3	5, 4	5, 5	5, 6	5, 7	5, 8	4, 9	5, 9	5,10	4,11	5,12	5,13	5,14
6, 1	6, 2	6, 3	6, 4	6, 5	6, 6	6, 7	7, 7	6, 8	6, 9	6,10	5,11	6,12	6,13	6,14
7, 1	0, 0	7, 2	7, 3	7, 5	7, 6	0, 0	0, 0	7, 8	7, 9	7,10	6,11	7,12	7,13	7,14
0, 0	0, 0	8, 2	8, 3	7, 4	8, 6	0, 0	0, 0	8, 8	8, 9	8,10	7,11	0, 0	0, 0	0, 0
8, 1	0, 0	0, 0	8, 4	8, 5	8, 7	0, 0	0, 0	9, 8	9, 9	9,10	8,11	8,12	8,13	8,14
9, 1	9, 2	9, 3	9, 4	9, 5	9, 6	9, 7	0, 0	10, 8	10, 9	10,10	9,11	9,12	9,13	9,14
10, 1	10, 2	10, 3	10, 4	10, 5	10, 6	10, 7	0, 0	0, 0	11, 9	11,10	10,11	10,12	10,13	10,14
11, 1	11, 2	11, 3	11, 4	11, 5	11, 6	11, 7	0, 0	0, 0	0, 0	0, 0	11,11	11,12	11,13	11,14
12, 1	12, 2	12, 3	12, 4	12, 5	12, 6	12, 7	12, 8	0, 0	12, 9	12,10	12,11	12,12	12,13	12,14
13, 1	13, 2	13, 3	13, 4	13, 5	13, 6	13, 7	0, 0	11, 8	13, 9	13,10	13,11	13,12	13,13	13,14
14, 1	14, 2	14, 3	14, 4	14, 5	14, 6	14, 7	14, 8	13, 8	14, 9	14,10	14,11	14,12	14,13	14,14

Figure .13: Move with distance 6.

1, 1	1, 2	1, 3	1, 4	1, 5	1, 6	1, 7	1, 8	0, 0	1, 9	1,10	1,11	1,12	1,13	1,14
2, 1	2, 2	2, 3	2, 4	2, 5	2, 6	2, 7	2, 8	0, 0	0, 0	0, 0	2,10	2,11	2,13	2,14
3, 1	3, 2	3, 3	3, 4	3, 5	3, 6	3, 7	3, 8	2, 9	0, 0	0, 0	3,10	2,12	3,13	3,14
4, 1	4, 2	4, 3	4, 4	4, 5	4, 6	4, 7	4, 8	3, 9	0, 0	4,10	3,11	3,12	4,13	4,14
5, 1	5, 2	5, 3	5, 4	5, 5	5, 6	5, 7	5, 8	4, 9	5, 9	5,10	4,11	4,12	5,13	5,14
6, 1	6, 2	6, 3	6, 4	6, 5	6, 6	6, 7	7, 7	6, 8	6, 9	6,10	5,11	5,12	6,13	6,14
7, 1	0, 0	7, 2	7, 3	7, 5	7, 6	0, 0	0, 0	7, 8	7, 9	7,10	6,11	6,12	7,13	7,14
0, 0	0, 0	8, 2	8, 3	7, 4	8, 6	0, 0	0, 0	8, 8	8, 9	8,10	7,11	7,12	0, 0	0, 0
8, 1	0, 0	0, 0	8, 4	8, 5	8, 7	0, 0	0, 0	9, 8	9, 9	9,10	8,11	8,12	8,13	8,14
9, 1	9, 2	9, 3	9, 4	9, 5	9, 6	9, 7	0, 0	10, 8	10, 9	10,10	9,11	9,12	9,13	9,14
10, 1	10, 2	10, 3	10, 4	10, 5	10, 6	10, 7	0, 0	0, 0	11, 9	11,10	10,11	10,12	10,13	10,14
11, 1	11, 2	11, 3	11, 4	11, 5	11, 6	11, 7	0, 0	0, 0	0, 0	0, 0	11,11	11,12	11,13	11,14
12, 1	12, 2	12, 3	12, 4	12, 5	12, 6	12, 7	12, 8	0, 0	12, 9	12,10	12,11	12,12	12,13	12,14
13, 1	13, 2	13, 3	13, 4	13, 5	13, 6	13, 7	0, 0	11, 8	13, 9	13,10	13,11	13,12	13,13	13,14
14, 1	14, 2	14, 3	14, 4	14, 5	14, 6	14, 7	14, 8	13, 8	14, 9	14,10	14,11	14,12	14,13	14,14

Figure .14: Move with distance 8.

1, 1	1, 2	1, 3	1, 4	1, 5	1, 6	1, 7	1, 8	0, 0	1, 9	1,10	1,11	1,12	1,13	1,14
2, 1	2, 2	2, 3	2, 4	2, 5	2, 6	2, 7	2, 8	0, 0	0, 0	0, 0	2,10	2,11	2,13	2,14
3, 1	3, 2	3, 3	3, 4	3, 5	3, 6	3, 7	3, 8	2, 9	0, 0	0, 0	3,10	2,12	3,13	3,14
4, 1	4, 2	4, 3	4, 4	4, 5	4, 6	4, 7	4, 8	3, 9	0, 0	4,10	3,11	3,12	4,13	4,14
5, 1	5, 2	5, 3	5, 4	5, 5	5, 6	5, 7	5, 8	4, 9	5, 9	5,10	4,11	4,12	5,13	5,14
6, 1	6, 2	6, 3	6, 4	6, 5	6, 6	6, 7	7, 7	6, 8	6, 9	6,10	5,11	5,12	6,13	6,14
7, 1	0, 0	7, 2	7, 3	7, 5	7, 6	0, 0	0, 0	7, 8	7, 9	7,10	6,11	6,12	7,13	7,14
0, 0	0, 0	8, 2	8, 3	7, 4	8, 6	0, 0	0, 0	8, 8	8, 9	8,10	7,11	7,12	8,13	0, 0
8, 1	0, 0	0, 0	8, 4	8, 5	8, 7	0, 0	0, 0	9, 8	9, 9	9,10	8,11	8,12	9,13	8,14
9, 1	9, 2	9, 3	9, 4	9, 5	9, 6	0, 0	0, 0	9, 7	10, 8	10, 9	10,10	9,11	9,12	9,14
10, 1	10, 2	10, 3	10, 4	10, 5	10, 6	10, 7	0, 0	0, 0	11, 9	11,10	10,11	10,12	10,13	10,14
11, 1	11, 2	11, 3	11, 4	11, 5	11, 6	11, 7	0, 0	0, 0	0, 0	0, 0	11,11	11,12	11,13	11,14
12, 1	12, 2	12, 3	12, 4	12, 5	12, 6	12, 7	12, 8	0, 0	12, 9	12,10	12,11	12,12	12,13	12,14
13, 1	13, 2	13, 3	13, 4	13, 5	13, 6	13, 7	0, 0	11, 8	13, 9	13,10	13,11	13,12	13,13	13,14
14, 1	14, 2	14, 3	14, 4	14, 5	14, 6	14, 7	14, 8	13, 8	14, 9	14,10	14,11	14,12	14,13	14,14

Figure .15: Move with distance 9.

1, 1	1, 2	1, 3	1, 4	1, 5	1, 6	1, 7	1, 8	0, 0	1, 9	0, 0	1,10	1,11	1,12	1,13
2, 1	2, 2	2, 3	2, 4	2, 5	2, 6	2, 7	2, 8	0, 0	0, 0	0, 0	2,10	2,11	2,13	1,14
3, 1	3, 2	3, 3	3, 4	3, 5	3, 6	3, 7	3, 8	2, 9	0, 0	0, 0	3,10	2,12	3,13	2,14
4, 1	4, 2	4, 3	4, 4	4, 5	4, 6	4, 7	4, 8	3, 9	0, 0	4,10	3,11	3,12	4,13	3,14
5, 1	5, 2	5, 3	5, 4	5, 5	5, 6	5, 7	5, 8	4, 9	5, 9	5,10	4,11	4,12	5,13	4,14
6, 1	6, 2	6, 3	6, 4	6, 5	6, 6	6, 7	7, 7	6, 8	6, 9	6,10	5,11	5,12	6,13	5,14
7, 1	0, 0	7, 2	7, 3	7, 5	7, 6	0, 0	0, 0	7, 8	7, 9	7,10	6,11	6,12	7,13	6,14
0, 0	0, 0	8, 2	8, 3	7, 4	8, 6	0, 0	0, 0	8, 8	8, 9	8,10	7,11	7,12	8,13	7,14
8, 1	0, 0	0, 0	8, 4	8, 5	8, 7	0, 0	0, 0	9, 8	9, 9	9,10	8,11	8,12	9,13	8,14
9, 1	9, 2	9, 3	9, 4	9, 5	9, 6	0, 0	0, 0	9, 7	10, 8	10, 9	10,10	9,11	9,12	9,14
10, 1	10, 2	10, 3	10, 4	10, 5	10, 6	10, 7	0, 0	0, 0	11, 9	11,10	10,11	10,12	10,13	10,14
11, 1	11, 2	11, 3	11, 4	11, 5	11, 6	11, 7	0, 0	0, 0	0, 0	0, 0	11,11	11,12	11,13	11,14
12, 1	12, 2	12, 3	12, 4	12, 5	12, 6	12, 7	12, 8	0, 0	12, 9	12,10	12,11	12,12	12,13	12,14
13, 1	13, 2	13, 3	13, 4	13, 5	13, 6	13, 7	0, 0	11, 8	13, 9	13,10	13,11	13,12	13,13	13,14
14, 1	14, 2	14, 3	14, 4	14, 5	14, 6	14, 7	14, 8	13, 8	14, 9	14,10	14,11	14,12	14,13	14,14

Figure .16: Move with distance 11.

Bibliography

[1] C. H. Stapper. Integrated circuit yield statistics. *Proceedings of the IEEE*, 71(4):453–470, April 1983.

[2] Paul R. Pukite and Claude L. Berman. Defect cluster analysis for wafer-scale integration. *IEEE Transactions on semiconductor manufacturing*, 3(3):128–135, August 1990.

[3] C. H. Stapper. Fault-simulation programs for integrated-circuit yield estimations. *IBM J. Res Develop*, 33(6):647–652, Nov 1989.

[4] Jung H. Kim and Phill K. Rhee. The rule-based approach to reconfiguration of 2-D processor arrays. *IEEE Transactions on Computers*, 42(11):1403–1408, November 1993.

[5] Bruno Ciciani. *Manufacturing Yield Evaluation of VLSI/WSI Systems.* IEEE Computer Society Press, 1995.

[6] C. H. Stapper, A. N. McLaren, and M. Dreckmann. Yield model for productivity optimization of VLSI memory chips with redundancy and partially good products. *IBM Journal on Research and Development*, 24(3):398–409, May 1980.

[7] C. H. Stapper. Improved yield model for fault-tolerant models memory chips. *IEEE Transactions on Computers*, 42(7):872–881, July 1993.

[8] C. H. Stapper. Large-area fault clusters and fault tolerance in VLSI circuits: A review. *IBM J. Res Develop*, 33(2):162–173, March 1989.

[9] Mengly Chean and Jose A. B. Fortes. A taxonomy of reconfiguration techniques for fault-tolerant processor arrays. *IEEE Computer*, pages 55–68, January 1990.

[10] Yung-Yuan Chen, Yung-Shiuan Shyu, and Ching-Hwa Cheng. An effective framework for fault-tolerant VLSI/WSI arrays based on hybrid redundancy approach. *IEEE International Conf. on Wafer Scale Integration*, pages 153–162, 1994.

[11] Adit D. Singh. Interstitial redundancy: An area efficient fault tolerance scheme for large area VLSI processor arrays. *IEEE Transactions on Computers*, 37(11):1398–1410, November 1988.

[12] T. Leighton and C. E. Leiserson. Wafer-scale integration of systolic arrays. *IEEE Trans. Computers*, C(34):448–461, May 1985.

[13] Mengly Chean and Jose A. B. Fortes. The full-use-of-suitable-spares (fuss) approach to hardware reconfiguration of fault tolerant processor arrays. *IEEE Transactions on Computers*, 39(4):564–571, April 1990.

[14] C. S. Rim J. Narasimhan, K. Nakajima and A. T. Dahbura. Yield enhancement of programmable ASIC arrays by reconfiguration of circuit placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(8):976–986, August 1994.

[15] Michal Cutler Minghsien Wang and Stephen Y. H. Su. Reconfiguration of VLSI/WSI mesh array processors with two-level redundancy. *IEEE Transactions on Computers*, 38(4):547–553, April 1989.

[16] Allan H. Anderson, J. I. Raffel, and P. W. Wyatt. Wafer scale integration using restructurable VLSI. *IEEE Computers*, pages 41–47, April 1992.

[17] Ahmed Boubekeur, Jean-Luc Patry, and Gabriele Saucier. Configuring a wafer-scale two-dimensional array of single-bit processors. *IEEE Computer*, pages 29–40, April 1992.

[18] T. E. Mangir and A. Avizienes. Fault-tolerant design for VLSI: Effects of interconnect requirements on yield improvement of VLSI designs. *IEEE Trans. Computer*, C(31):609–615, July 1982.

[19] J. Bernard. The IC yield problem: A tentative yield analysis for MOS/SOS circuits. *IEEE Transactions on Electronic Devices*, ED-25:939–944, August 1978.

[20] Phill K. Rhee and Jung H. Kim. Channel complexity analysis for reconfigurable VLSI/WSI processor arrays. *International Conference on Application Specific Array Processors*, pages 329–340, 1990.

[21] Phill K. Rhee. *On the Reconfiguration of VLSI Processor Arrays.* PHD dissertation,Center of Advanced Computer Studies, University of Southwestern Louisiana, 1990.

[22] M. G. Sami Fabrizio Lombardi and R. Stefanelli. Reconfiguration of VLSI arrays by covering. *IEEE Transactions on Computers*, 8(9):952–965, September 1989.

[23] Shann-Ning Jean Sun-Yuan Kung and Chih-Wei Chang. Fault-tolerant array processors using single-track switches. *IEEE Trans. Computers*, 38(4):500–514, April 1989.

[24] Sy-Yen Kuo and W. K. Fuchs. Efficient spare allocation for reconfigurable arrays. *IEEE Design and Test*, pages 24–31, Feburary 1987.

[25] Isreal Koren and Adit D. Singh. Fault tolerance in VLSI circuits. *IEEE computers*, pages 73–83, July 1990.

[26] James A. Cunningham. The use and evaluation of yield models in integrated circuit manufacturing. *IEEE Transactions on Semiconductor Manufacturing*, 3(2):61–71, May 1990.

[27] Wojciech Maly, Andrzej J. Strojwas, and Stephen W. Director. VLSI yield estimation and prediction : A unified framework. *IEEE Transactions on Computer Aided Design*, CAD-5(1):114–130, January 1986.

[28] Wojciech Maly. Computer aided design for VLSI circuit manufacturability. *Proceedings IEEE*, 78(2):356–392, Feburary 1990.

[29] I. Koren and C. H. Stapper. Yield models for defect-tolerant VLSI circuits : A review. *Defects and Fault Tolerance in VLSI Systems*, 1(2):1–21, June 1988.

[30] T. L. Michalka, Ramesh C. Varshney, and J. D. Meindl. A discussion of yield modeling with defect clustering, circuit repair, and circuit redundancy. *IEEE Transaction on Semiconductor Manufacturing*, 3(3):116–127, August 1990.

[31] C. H. Stapper. Small-area fault clusters and fault tolerance in VLSI circuits. *IBM J. Res Develop*, 33(2):174–177, March 1989.

[32] Z. Koren I. Koren and C. H. Stapper. A unified negative-binomial distribution for yield analysis of defect tolerant circuits. *IEEE Transactions on Computers*, 42(6):724–733, June 1993.

[33] Z. Koren I. Koren and C. H. Stapper. A statistical study of defect maps of large area VLSI IC's. *IEEE Transactions on Very Large Scale Integration Systems*, 2(2):249–256, June 1994.

[34] Albert V. Ferris-Prabhu. A cluster-modified poisson model for estimating defect density and yield. *IEEE Trans. Semiconductor Manufacturing*, 3(2):54–59, May 1990.

[35] Mark B. Ketchen. Point defect yield model for wafer scale integration. *IEEE Circuits and Devices Magazine*, pages 24–34, 1985.

[36] Averill M Law and W. David Kelton. *Simulation Modelling and Analysis.* McGraw Hill Inc., 1991.

[37] A. V. Ferris-Prabhu et. al. Radial yield variations in semiconductor wafers. *IEEE Circuits and Devices Magazine*, pages 42–47, March 1987.

[38] F. J. Meyer and D. K. Pardhan. Modeling defect spatial distribution. *IEEE Transactions on Computers*, 38(4):538–546, April 1989.

[39] Yung-Yuan Chen, Ching-Hwa Cheng, and Yung-Ci Chou. The design and analysis of fault-tolerant VLSI/WSI array processors. pages 637–641.

[40] Bruno Codennotti and Roberto Tamassia. A network flow approach to the reconfiguration of VLSI arrays. *IEEE Transactions on Computers*, 40(1):118–121, January 1991.

[41] I. Takanami, Y. Hisanaga, and K. Inoue. Switching netwroks and neural algorithms for reconstructing mesh-connected processor arrays with spares on their sides. pages 360–365, May 1994.

[42] Kuochen Wang and Jenn-Wei Lin. Integrated diagnosis and reconfiguration process for defect tolerant WSI Processor Arrays. *IEEE International Conference on Wafer Scale Integration*, pages 198–207, 1994.

[43] Laurence E. LaForge. What designers of wafer scale systems should know about local sparing. *IEEE International Conference on Wafer Scale Integration*, pages 106–131, 1994.

[44] N. J. Davis et. al. Reconfiguring fault-tolerant two-dimensional array architectures. *IEEE Micro*, pages 60–69, April 1994.

[45] Thanos Stouraitis. Borrow: A fault-tolerance scheme for wavefront array processors. *IEEE Transactions on Computers*, 42(10):1257–1261, October 1993.

[46] Robert W. Horst. Task-flow architecture for WSI parallel processing. *IEEE Computers*, pages 10–18, April 1992.

[47] Vwani P. Rowchowdhury Theodora A. Varvarigou and Thomas Kailath. A polynomial time algorithm for reconfiguring multiple-track models. *IEEE Transactions on Computers*, 42(4):385–395, April 1993.

[48] S. Horiguchi and Issei Numata. A self reconfiguration architecture for mesh arrays. *IEEE International Workshop on Defect and Fault Tolerance in VLSI Systems*, pages 212–220, 1994.

[49] Shann-Ning Jean, H. C. Fu, and Sun-Yuan Kung. Yield enhancement for WSI array processors using two-and-half-track switches. *IEEE Int'l Conf. on WSI*, pages 243–250, 1990.

[50] Vwani P. Rowchowdhury Theodora A. Varvarigou and Thomas Kailath. Reconfiguring processor arrays using multiple-track models: The 3-track-1-spare-approach. *IEEE Transactions on Computers*, 42(11):1281–1293, November 1993.

[51] M. Tarr, D. Boudreau, and R. Murphy. Defect analysis system speeds test and repair of redundant memories. *Electronics*, pages 175–179, January 1984.

[52] J. Day. A fault-driven comprehensive redundancy algorithm. *IEEE Design and Test*, 2(3):35–44, June 1985.

[53] Michael D. Smith and Pinaki Mazumder. Generation of minimal vertex covers for row/column allocation in self-repairable arrays. *IEEE Transactions on Computers*, 45(1):109–114, January 1996.

[54] R. Negrini M. G. Sami and R. Stefanelli. *Fault Tolerance Through Reconfiguration in VLSI and WSI Arrays.* The MIT Press, Cambridge, Massachusetts, London, England, 1989.

[55] S Micali and V. Vazzurani. *An* $O\left(\sqrt{|V|} \times |E|\right)$ *algorithm for finding maximum matching in general graphs.* Proc. Symp. Found. Comput. Sci. pg 17-27, 1980.

[56] Yung-Yuan Chen and Shambhu J. Upadhyaya. Yield analysis of reconfigurable processors based on multiple-level redundancy. *IEEE Transactions on Computers*, 42(9):1136–1141, September 1993.

[57] Yung-Yuan Chen, Sau-Gee Chen, and Jiann-Cherng Lee. Yield and performance issues in fault-tolerant WSI array architectures. *IEEE International Conf. on Wafer Scale Integration*, pages 319–328, 1995.

[58] T. Liu and F. Lombardi. On soft switch programming for reconfigurable array systems. *International Workshop on Defect and Fault Tolerance in VLSI Systems*, pages 203–211, 1994.

VITA

- Syed Shah Hadi Hussain Qadri
- Born in 1971 at Hyderabad, India.
- Permanent Address :

  12-13-483/A/1, Street 1, Tarnaka, Hyderabad. India. Pin 500 017
- Received Bachelor of Engineering (**B.E.**) degree in Electronics and Communication Engineering from Osmania University, Hyderabad, India in June 1992.
- Received Master of Science (**M.S.**) degree in Computer Engineering, in June 1996.